NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IPV6 HOST FINGERPRINT

by

Eleftherios Nerakis

September 2006

Thesis Advisor:	Geoffrey Xie
Co-Advisor	John Gibson
Second Reader:	Chris Eagle

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE IPV6 Host Fingerprint			**5. FUNDING NUMBERS**
6. AUTHOR(S) Eleftherios Nerakis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis explores ways of using probe packets to identify the type and version of OS that is run by a remote IPv6 host. Such a probing technique can be effective because developers of different OSes often interpret the guidance provided by the RFCs slightly differently, and consequently their network protocol stack implementation may generate responses bearing unique markers to certain probing packets. The key challenge is to find suitable probing packets for different OSes. Using a real IPv6 test bed, this thesis has evaluated both existing UDP-or-TCP-based and new IPv6-extension-header-based probing packets against a selected set of eight popular OSes. The results show that the UDP/TCP methods are also effective in an IPv6 environment and the extension header approach is worthy further study. There are evidences that OS fingerprinting is harder with IPv6. It might be due to the fact that given the experimental nature of IPv6, similar OSes tend to reuse IPv6 code. This conjecture requires further study. Finally, the thesis has also developed a method of crafting arbitrary IPv6 packets using the SmartBits system.

14. SUBJECT TERMS IPv6, fingerprint, OS detection			**15. NUMBER OF PAGES** 119
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IPV6 HOST FINGERPRINT

Eleftherios Nerakis
Lieutenant, Hellenic Navy
B.S. Hellenic Naval Academy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author: Eleftherios Nerakis

Approved by: Dr. Geoffrey Xie
Thesis Advisor

John Gibson
Co-Advisor

Chris Eagle
Second Reader

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis explores ways of using probe packets to identify the type and version of OS that is run by a remote IPv6 host. Such a probing technique can be effective because developers of different OSes often interpret the guidance provided by the RFCs slightly differently, and consequently their network protocol stack implementation may generate responses bearing unique markers to certain probing packets. The key challenge is to find suitable probing packets for different OSes. Using a real IPv6 test bed, this thesis has evaluated both existing UDP-or-TCP-based and new IPv6-extension-header-based probing packets against a selected set of eight popular OSes. The results show that the UDP/TCP methods are also effective in an IPv6 environment and the extension header approach is worthy further study. There are evidences that OS fingerprinting is harder with IPv6. It might be due to the fact that given the experimental nature of IPv6, similar OSes tend to reuse IPv6 code. This conjecture requires further study. Finally, the thesis has also developed a method of crafting arbitrary IPv6 packets using the SmartBits system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

I. INTRODUCTION

A. IMPORTANCE OF OS DETECTION

The Internet today, as it is defined in the book, *Computer Networking—A Top Down Approach*" by Kurose-Ross, is a "network of networks" [1]. This worldwide network consists of end systems, either application clients or servers, and the communication infrastructure that interconnects them. This worldwide network provides the digital highway for the wide range of services that are available today, and makes the Internet an attractive tool with a vast amount of users. Although this network was built to provide services to users with goodwill, it has become a target of malicious attacks.

This is the point where the concept of network security arises with the ultimate aim to protect the assets of this network. The assets in the Internet could be either the devices that make up the Internet or the data that are stored in those devices. Because the Internet initially was not built with great concern for security, over the years it has evolved to eliminate the vulnerabilities and provide its users the appropriate level of assurance. Similarly, the end systems have been under the same evolution, and the effort was to develop operating systems (OSes) that would be self-protected and tamper-proof. Nevertheless, vulnerabilities still exist in the systems today and may still exist in the future No matter how hard it is to identify vulnerabilities, there are people devoted to reveal them and either to correct the problem by performing the appropriate modifications on the software or to exploit them by staging attacks.

All computer systems today run an operating system, and there are a variety of OSes from which to choose. Because these OSes have been developed by different vendors, they have been implemented according to their developers' own philosophy about how to best provide assurance to their users and protect the data they store. Until the time when an absolutely secure system with no vulnerability is developed, all OSes will have vulnerable points with the potential to be exploited. Those vulnerabilities are, in fact, weak points in the software of the OS, and this is the reason that makes vulnerabilities OS specific. Different OSes will have different vulnerabilities, and if someone wishes to take advantage of them, he will probably need to follow a different

approach depending on the OS being attacked. Moreover, the OS is the software platform for several other applications, which may also have their own vulnerabilities that can be exploited to take control of a system or perform malicious actions on that system.

So, we come to the conclusion that the knowledge of the OS type and the services running on a system are the key factors for deciding the appropriate methods for attacking a given system. To date, a variety of techniques have been explored to remotely identify the OS type over the Internet and a number of tools have been developed to automate the process and make it much easier. However, the ability to identify the OS can also be used for defensive purposes. Network administrators should be aware of the importance of OS detection and take all the necessary measures to frustrate this threat. Thus, one way to verify the reliability of the protection measures they have taken is to use the same tools against their own network and determine what kind of information is leaving their network that could reveal the OSes running on their machines. These tools most often use a database of common characteristics associated with specific OSes. That means a database must be developed in advance with which to compare the observed characteristics of an OS and then to deduce the type being used. Those characteristics are pretty much like fingerprints for OSes and thus the overall process can be defined as OS fingerprinting.

B. THESIS OBJECTIVES

The Internet today implements the IP version 4 protocol within the network layer of the protocol stack, as it is presented in Figure 1. That means it uses the Internet protocol of the network layer, as it is defined in RFC 791. This standard describes the datagram format, the addressing conventions, and the packet handling conventions of the packets or datagrams traversing the network layer.

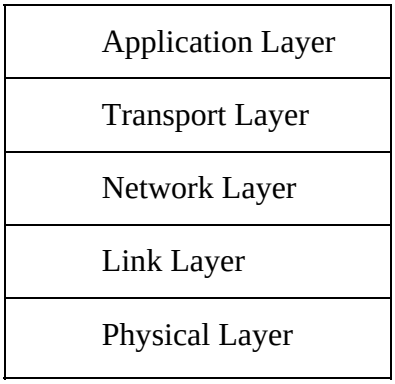

Figure 1. TCP/IP protocol stack.

The most important characteristic of the TCP/IPv4 protocol stack is the addressing part. Today, all hosts connected to the Internet, in order to be able to send and receive datagrams, need to have an IP address. Each IP address is 32 bits long and is unique for every interface that connects a host to the Internet. This architecture works very well and provides a great degree of efficiency. However, difficulties arise from the fact that the address space defined with the existing 32-bit addressing architecture has an upper bound. It can address no more than 4.2 billion hosts[1]. Although, this is a relatively large number of hosts, with the current rate of new users attaching to the Internet, it has been estimated by two leaders of the IETF Address Lifetime Expectations working group that addresses would become exhausted in 2008 and 2018, respectively [2,3]. Clearly something has to be done about this. Thus, in the early '90s the IETF began an effort to develop the successor of the IPv4 protocol. The solution is the IPv6 protocol [4], which probably solves the address space limitation once and for all but also introduces a few more changes based on experience gained from IPv4.

Based on the fact that a transition from IPv4 to IPv6 will occur in the next several years, a number of questions arise regarding the applicability of OS fingerprinting to IPv6. Thus, the main objectives for this thesis are the following:

- Investigate the applicability of the techniques used currently for the IPv4 protocol to the forthcoming IPv6.

[1] $2^{32} \cong 4.2$ billions

- The IPv6 protocol may enable new methods of OS fingerprinting. A secondary goal of this thesis is to identify some of these methods.

C. THESIS OVERVIEW

This section presents briefly the contents of the subsequent chapters.

1. Chapter II, Background

This chapter describes the approaches available today for detecting the OS of a remote host in an IPv4 environment and presents the available methods for active stack fingerprinting. The objective of the information presented in this chapter is to help the reader create a solid base around the concept of OS detection and especially of OS fingerprinting.

2. Chapter III, Network Configuration and Packet Crafting

In this chapter, the experimentation network, which was set up for the purpose of the thesis, is presented. The factors on which the selection of the types and the vendors of the OSes included in the network were based are also described. Finally, it describes the packet crafting process and how it was conducted in this thesis.

3. Chapter IV, Testing Of Existing IPv4-Based Methods

Two tools available today for OS detection, Nmap and Queso, were explored and used for OS detection in the test network. Nmap seems to use the most efficient and complete methods for OS fingerprinting. The results of using those tools with the network, in order to have a baseline of OS detection in IPv4 environment, are described. Then, these methods are tested for their applicability with IPv6. The results of the implementation of each method on the selected OSes in the network are presented in a table format and analyzed in order to find identifying factors among the OSes.

4. Chapter V, OS Detection Methods Enabled by IPv6

This chapter reports an attempt to identify new OS detection methods enabled by the IPv6 protocol itself. New methods are possible because a lot of changes were made to the IP architecture, and it is likely that new identifying factors may be found in the implementation of the new protocol by different OSes.

5. Chapter V, Conclusions

Finally, the results from the tests are summarized and a discussion is presented about the effectiveness of those methods into IPv6. This chapter also discusses possible ways to extend this research

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter will provide the reader with a solid base with respect to the OS fingerprinting concept and explore the approaches taken so far to detect the OS resident on a system of interest. This is important because, even though there is to be a transition from IPv4 to IPv6, it is possible that the same concepts used for OS fingerprinting in the first protocol can be used in the second as well.

A. TYPES OF OS DETECTION

Several different approaches have been developed for detecting the OS of a host. Each one of them has unique pros and cons, and its effectiveness varies from OS to OS. The basic approaches used today are briefly described below. However, they could be categorized into four classes based on the type of information they use to identify the OS. The first two are looking for information gathered at the application layer level, such as which ports are open on a host or any explicit reference from the application about the name of the application or the developer. The next two use the concept of "stack fingerprinting"[5,6], which looks more deeply into the packets sent from the target host and attempts to determine the OS by examining the values used for some of the fields inside the various layer headers. Because a specific set of values could identify the OS, like a specific fingerprint identifies a person, we refer to this set as an OS fingerprint.

1. Port Scanning

Port scanning is the process of probing the TCP and UDP ports on a target host to identify the services that are running and listening for incoming connections. Although this is not explicitly an OS detection technique, it can be used to determine the type and version of the OS [7]. This is because many services are known to run on specific types of OS. For example, the NetBIOS name service of the Windows OS listens on UDP port 137. That means that if we discover this port open on a host, it most probably is running a Windows OS. However, this technique is not always successful, as many services are capable of running on different OS types. For example the Web service uses the HTTP application layer protocol, which is identified with the well-known port 80. This port is

the same for every host used as a public web server. In this case, identifying that port 80 is open and listening for incoming connections does not say too much about the type of OS on the host.

Some of the port scanning techniques used today are described later when we discuss the OS detection methods. Because most of these techniques are used for Active Stack OS detection, the difference lies in the way we manipulate the results found. In the case of Port Scanning, we are looking for active processes. So, if we attempt to establish a TCP connection with the target host by sending a SYN packet, then that machine will send back either a RST/ACK packet if the port is closed or a SYN/ACK packet[2] if the port is open. If we send this packet in sequence to all ports on the target host, we can discover all the open and closed ports on that machine[3].

2. Banner Grabbing

With banner grabbing, we simply try to connect to applications running on the target host and observe the responses, which sometimes may reveal very useful information as to the exact type and developer of the application [8]. However, this is not always effective. The type of information leaving a host is often configurable and many administrators have taken care of this when setting up their network. So, this type of OS detection is most effective only in the case of a misconfigured server.

3. Passive Stack Fingerprinting

Passive stack fingerprinting [9] attempts to identify the OS by monitoring network traffic and examining the values set for some of the fields of the packets sent from the target host. Some of these fields are the time to live (TTL), window size, initial sequence number (ISN), the don't fragment (DF), and possibly others. Passive stack fingerprinting will identify the values set for these fields and will compare them with a database and then infer the OS based on the similarities. This means that we must have a database developed in advance that catalogs OSes by a set of values for those attributes. This approach, although it can be performed silently without leaving any trace, has a major limitation in that we must be inside the network for which we want to see the traffic.

[2] This is the TCPconnect scan used by Nmap, where we are trying to attempt a full three-way handshake with the target host.

[3] The port number is 16 bits long, so there are $2^{16} = 65535$ different ports.

4. Active Stack Fingerprinting

The most effective method investigated by the author is active stack fingerprinting [10]. This approach is based on the observation that there are cases where some vendors interpret or implement specific RFC guidance differently from other vendors when they develop their TCP/IP protocol stack. So, by probing for these differences we can come up with a very good conjecture as to the exact type of OS in use. As with the passive stack fingerprinting, these differences in most cases are different values that are set by the OS in some of the fields of the packets they send out. Those values are compared with a database, as with passive stack fingerprinting, to identify the likely implementation. Also, it is possible to observe different behavior by different OSes in response to the same probing methods.

One might ask at this point why different OSes use different values or display different behavior if they conform to the same standard. In the Internet, the format, syntax, and sequence of all packets exchanged between the communicating hosts are defined in a great detail by the well-known RFCs. However, there are cases where these standards provide the OS's vendor with a degree of flexibility to use values for some of the fields in the headers of the protocol stack. Further, the vendors may interpret differently the guidance provided from the RFCs during the development of their protocol stack. Although this is not a major problem, if it is one at all, for the end systems to communicate over the Internet, it may help to reveal the underlying OS.

There are many methods used today to probe for these differences because the more methods used, the more differences that may be found and so the guess would be more accurate. Another reason is because of security implementations. Administrators may block specific types of packets from entering their network, most often by establishing a firewall and configuring it appropriately. Innovative, out-of-the-box thinking may lead to the creation of packets that can circumvent those restrictions and reach the intended destination machine.

B. METHODS OF ACTIVE STACK FINGERPRINTING

Active stack fingerprinting is based on the fact that different OSes may respond differently when they are triggered in the same way. The key for the effectiveness of this technique is to find the appropriate packets that can probe for these differences. This is a process that demands a lot of testing and inspection of the RFC's guidance to detect points that could be interpreted differently by different vendors.The literature describes many methods that could probe for those differences. Active stack fingerprinting defines a specially constructed packet to trigger the target host to send back a response, which will include values that could point to a particular OS or constrained set of OSes. Specific methods found in the literature [10, 11] are described below and some of them are implemented by tools available today. These techniques use a combination of TCP header and IPv4 header field values to characterize the target OS.

1. FIN Probe

In this method, a FIN packet is sent to an open port. Although the common response, as it is directed by the RFC 793, is not to respond, there are OSes that send a response back with the FIN and ACK flags set.

2. Bogus Flag Probe

This method sends a packet with the SYN and an undefined flag bit set. Some OSes, such as Linux, will respond with the flag set in their response packet.

3. Initial Sequence Number

This method sends a series of connection request packets to the target machine, and from the responses we get back we record the initial sequence number (ISN) and we try to find a pattern in the selection of the ISN.

4. Don't Fragment Bit (DF)

This method observes the responses coming back from the target machine and monitors the DF bit in the IP header. Some OSes set this bit in order to enhance performance, but there are others that do not set that bit or set it only in specific cases.

5. Initial Window Size

This method monitors the initial window size value set by the target machine. This value can be unique for some OSes and thus can identify the OS.

6. ACK Value Probe

This method monitors the ACK value set by the target machine. There are cases where the value will be the sequence number we sent or the sequence number +1.

7. Type of Service (TOS)

In this method the type of service field of the ICMP "port unreachable" message sent back is examined. Most OSes use "0" but this may vary. There are cases, for example Linux, where they use 0xC0.

8. Fragmentation Handling

It is possible that different protocol stack implementations handle overlapping fragments differently. Some OSes will overwrite the old data with the new data, and vice versa, when they reassemble the fragments [12].

9. TCP Options

Although the supported TCP options are defined in RFC 793 and RFC 1323, it is possible that some OSes do not support all of them. Thus, by sending a packet with one or more options set, we can identify which options are supported by the target OS. Also, the supported options may be listed in a different order in the response, depending on the OS.

10. ICMP Error Message Quenching

RFC 1812 suggests that an OS should limit the rate at which it sends error messages. This can be tested by sending UDP packets to a closed port and then count the number of unreachable messages received within a given amount of time. This method has the disadvantage that some UDP packets may be dropped in the network, making it hard to compute accurate results.

11. ICMP Message Quoting

ICMP error messages should quote some amount of information from the packet that generated the ICMP error message. However, not all OSes quote the same amount of information. Thus, by monitoring the amount of quoted information it is possible to make a guess about the OS employed on the target.

12. ICMP Error Message-Echoing Integrity

As described in the paragraph above, when the OS sends back an ICMP error message, it quotes some amount of the original message received. Also, some stack

implementations change the IP headers of the original message. So, by examining the type of alterations made by the target, it is possible to make an assumption about the OS.

This chapter provided a survey of methods currently used to extract information either directly from target host configurations, such as port scans, or from the packets generated by those systems and sent over the network. The latter, referred to collectively as stack fingerprinting because of its extraction of information from various protocol layer headers, may be either passive or active. Some of the methods introduced here will be employed in a controlled network environment as described in the next chapter initially over the IPv4 and then over IPv6.

III. NETWORK CONFIGURATION

A. NETWORK SETUP

To identify and test the methods of OS fingerprinting used in a IPv4 environment, we need to set up a small network lab consisting of different OSes. Also, the same network will be used later when we test those methods in IPv6, thus the OSes of this network should be configured for dual protocol stack capability. This chapter first discusses the considerations taken into account for selecting the OSes included in the network, then describes the network architecture, and finally presents the packet crafting process, which is the core part of OS fingerprinting.

1. OS Selection

For the selection of the types of the OSes we will include in our network, we should take into consideration the number of OSes, the vendor of the OS, and the popularity of the OS. In order to have results as accurate as possible, we need to set up as many OSes as possible. Also, because the methods used today are looking for differences in the protocol stack implemented by different OSes developed by different vendors, it would by wise to include OSes from as many vendors as possible in our selection. And finally, we should use OSes that seem to have some level of popularity. Often, surveys about the popularity of the OSes are conducted . Two of them have been found over the Internet and they present very interesting information on this subject. Table 1 presents statistical data from a survey conducted by Statemarket.com [14] in a 3-month period back in 1999. Table 2 [15] also presents statistical data on the popularity of OSes. It covers a more recent period of time. From the analysis of this information, we conclude that Windows is almost the dominant OS today. However we observe that the popularity of others OSes like Linux and MacOS has increased during recent years and will probably continue to increase. However, the results may not be as realistic as possible, but the point is that the percentage of popularity for each of them is so distinct among the vendors that there is no doubt about the order of popularity.

Rank	OS	Avg.	Min. (Date)	Max. (Date)
1.	Windows 95	48.28%	41.93% (5/23/99)	54.23% (1/8/99)
2.	Windows 98	39.80%	33.29% (1/8/99)	48.17% (5/17/99)
3.	Windows NT	4.81%	2.86% (1/17/99)	6.40% (5/21/99)
4.	Macintosh	2.72%	2.40% (1/17/99)	3.04% (1/27/99)
5.	WebTV	1.81%	1.52% (3/17/99)	2.23% (1/13/99)
6.	Windows 3.*	1.38%	1.04% (5/23/99)	1.80% (1/13/99)
7.	Other	0.56%	0.09% (2/22/99)	0.69% (5/17/99)
8.	Linux	0.20%	0.16% (1/9/99)	0.22% (5/22/99)
9.	SunOS	0.16%	0.08% (5/10/99)	0.23% (2/19/99)
10.	Irix	0.04%	0.01% (3/2/99)	0.06% (2/24/99)

Table 1. OS statistics 1[14]

2006	WinXP	W2000	Win98	WinNT	W2003	Linux	Mac
June	74.1%	10.6%	1.6%	0.3%	2.0%	4.4%	3.6%
May	74.2%	10.7%	1.6%	0.2%	2.0%	3.4%	3.6%
April	74.0%	11.2%	1.8%	0.3%	1.9%	3.3%	3.6%
March	72.9%	11.9%	2.0%	0.3%	1.8%	3.4%	3.5%
February	73.3%	12.3%	2.1%	0.3%	1.8%	3.4%	3.6%
January	72.3%	13.1%	2.4%	0.3%	1.7%	3.3%	3.5%
2005	**WinXP**	**W2000**	**Win98**	**WinNT**	**W2003**	**Linux**	**Mac**
December	71.6%	13.6%	2.6%	0.3%	1.7%	3.2%	3.3%
November	71.0%	14.6%	2.7%	0.4%	1.7%	3.3%	3.3%
October	70.2%	15.0%	2.8%	0.4%	1.6%	3.3%	3.2%
September	69.2%	15.8%	3.2%	0.5%	1.7%	3.3%	3.1%
August	66.3%	17.5%	3.2%	0.6%	1.7%	3.3%	2.9%
July	65.3%	17.7%	3.9%	0.6%	1.6%	3.5%	3.0%
June	64.9%	19.1%	3.6%	0.7%	1.5%	3.5%	3.0%
May	64.5%	19.4%	3.9%	0.8%	1.4%	3.3%	2.9%
April	64.0%	19.7%	4.1%	0.8%	1.4%	3.3%	2.9%
March	63.1%	20.2%	4.7%	0.9%	1.4%	3.2%	3.0%
February	62.0%	21.1%	5.1%	0.9%	1.3%	3.2%	2.9%
January	61.3%	21.6%	5.3%	1.0%	1.2%	3.2%	2.8%
2004	**WinXP**	**W2000**	**Win98**	**WinNT**	**Win95**	**Linux**	**Mac**
December	59.8%	23.5%	5.4%	1.1%	0.1%	3.1%	2.7%
November	59.1%	23.7%	5.6%	1.2%	0.1%	3.1%	2.7%
October	57.8%	25.0%	6.0%	1.3%	0.2%	3.1%	2.6%
September	55.9%	26.2%	6.4%	1.5%	0.2%	3.1%	2.6%
August	53.2%	28.1%	7.0%	1.8%	0.2%	3.0%	2.5%
July	52.5%	28.4%	7.5%	1.9%	0.2%	3.1%	2.4%
June	51.2%	29.6%	8.0%	2.0%	0.3%	2.9%	2.5%
May	51.0%	29.6%	8.2%	2.0%	0.3%	2.9%	2.5%
April	49.7%	30.2%	8.7%	2.2%	0.3%	2.7%	2.5%
March	48.0%	31.1%	9.4%	2.4%	0.4%	2.6%	2.4%
February	46.0%	32.8%	9.5%	2.9%	0.4%	2.6%	2.5%
January	44.1%	33.6%	10.4%	3.0%	0.4%	2.7%	2.4%
2003	**WinXP**	**W2000**	**Win98**	**WinNT**	**Win95**	**Linux**	**Mac**
December	43.6%	35.2%	10.5%	3.4%	0.4%	2.7%	2.3%
November	42.6%	36.3%	10.9%	3.5%	0.4%	2.6%	2.2%
October	39.4%	37.8%	11.5%	4.0%	0.5%	2.5%	2.1%
September	38.0%	37.9%	12.1%	4.1%	0.5%	2.4%	2.0%
August	36.3%	39.9%	12.6%	4.6%	0.5%	2.4%	2.0%
July	33.9%	40.6%	12.6%	5.3%	0.6%	2.3%	1.9%
June	32.8%	40.4%	13.4%	5.4%	0.6%	2.3%	1.8%
May	31.4%	41.0%	13.9%	5.8%	0.7%	2.2%	1.8%
April	30.8%	40.9%	14.7%	6.0%	0.7%	2.1%	1.8%
March	29.1%	41.9%	14.8%	6.6%	0.8%	2.2%	1.8%

Table 2. OS statistics 2 [15]

Taking into account the above considerations we came up with the following list of OSes for our network:

- Windows Server 2003 Enterprise Edition
- Windows Server 2003 Standard Edition

- Windows XP Professional
- Red Hat Linux Enterprise 4 WS
- Red Hat Linux 9.0
- Fedora Core 4
- Mac OS X 10.4.5
- FreeBSD 6.0

2. Network Architecture

For our purposes, the network could be as simple as possible. The actual topology used for this thesis is depicted in Figure 2 below. It is composed of one hub, which interconnects all the hosts into one collision domain. In this network, there are four machines, in green color, running the different types of OSes. The yellow box represents the packet generator, for which we use a SmartBits 6000C device manufactured by Spirent Communications, Inc. The detailed functionalities of the SmartBits device and its setup for this thesis will be presented in the next section. The laptop above the “Smart Bits Chassis is necessary for controlling the packet generator. Finally, the packet analyzer is represented in blue. This is a simple PC, which runs a packet analyzer like Wireshark to capture the packets exchanged across the network.

Initially, the OSes were installed on the machines of the network, which is a challenging and time consuming process because of the multi-boot configuration. Next, each network module on every OS needs to be configured for dual stack network connectivity. This is necessary because we need to test the OS detection process first on IPv4 and later on IPv6. Finally, the SmartBits chassis is connected to the network and the “controller” and configured for its connectivity.

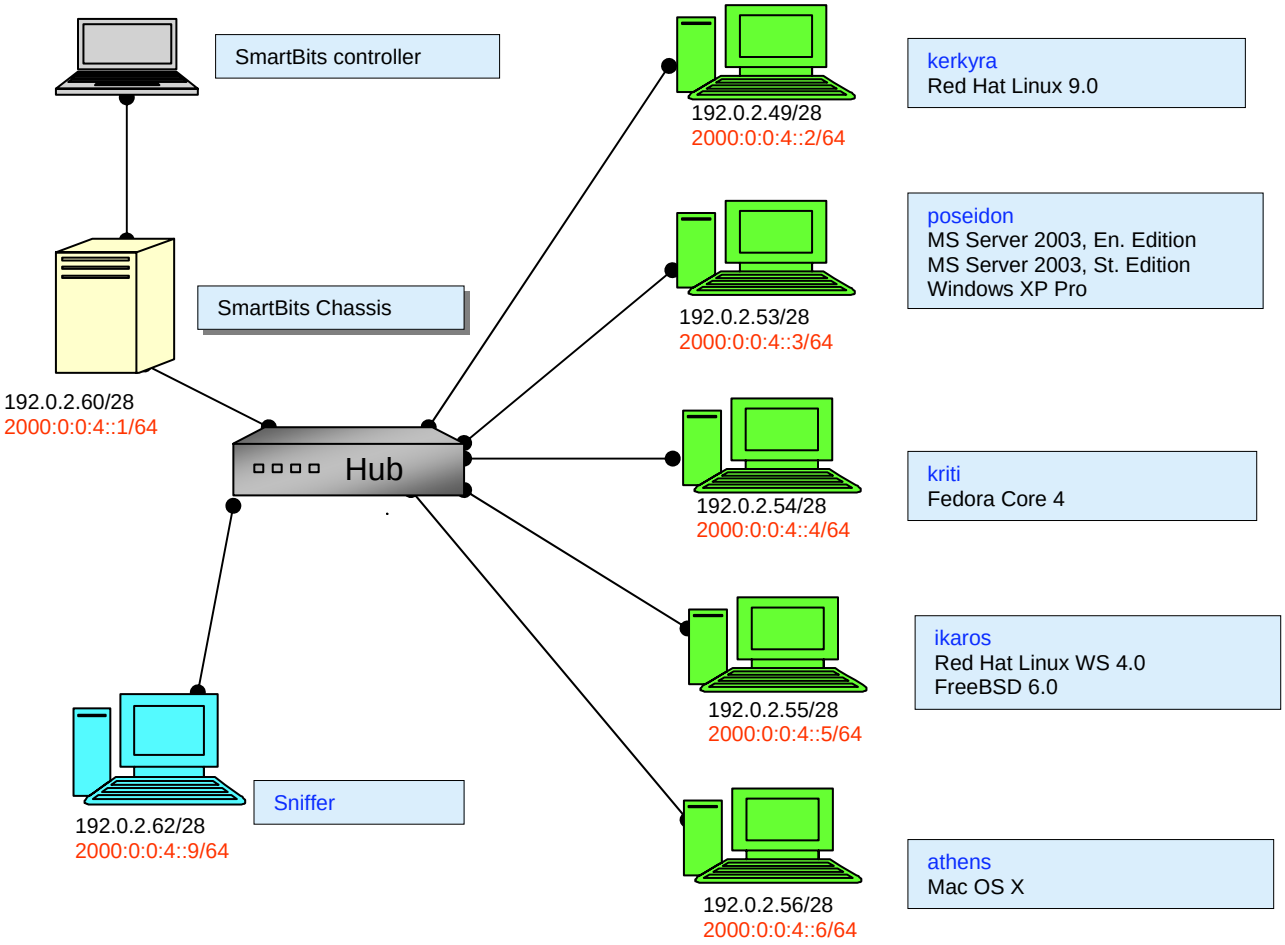

Figure 2. Network architecture.

B. PACKET CRAFTING

Central to the concept of OS fingerprinting is the generation of various packets that will trigger the remote host to respond in a way that will reveal some unique characteristics of the underlying OS. Any OS with a network module in its architecture supports the generation of packets that enable the applications running on top of it to communicate over the Internet. Those packets are, in most of the cases, ordinary packets exchanged across the network and are responsible for making the communication among the applications feasible. For OS detection, however, those types of packets may not be sufficient to make an accurate guess about the OS type. Here is where packet crafting may help to circumvent any of those restrictions. Packet crafting uses the same network protocol stack as any other application but gives the user the ability to form any kind of packet desired and send it to another host connected to the network.

Many tools are available today for packet crafting. Those tools normally support two modes of operation. One mode provides the user the ability to select the type of packet to be sent from a limited number of supported protocols, and the other mode gives the user the opportunity to create a packet by inserting the values of his choice in hexadecimal format and actually create a complete frame from the bottom up.

The last mode of operation, although it is very powerful, has the disadvantage that the user must be aware of the details of the protocols that he is about to use. This is because a single incorrect value may cause the packet to be interpreted as corrupted from the receiver and finally dropped without further notice to the sender. Also, it is very time consuming because all values, even the default, must be inserted manually and in the correct order. Finally, the calculation of the checksum, which is used by various protocols, must be accomplished. In this case, the user must calculate the checksum manually and then insert that value where ever is applicable.

1. SmartBits Overview

SmartBits is a tool that supports packet crafting, along with many more services, and was selected for use for this thesis. It is a performance analysis system that allows the user to test, simulate, analyze, troubleshoot, develop, and certify equipment such as routers, repeaters, bridges, and network interface cards (NICs), as well as VLANs, ELANs, and live networks. It is composed of the SmartBits hardware, shown in Figure 3 below, and the SmartBits software.

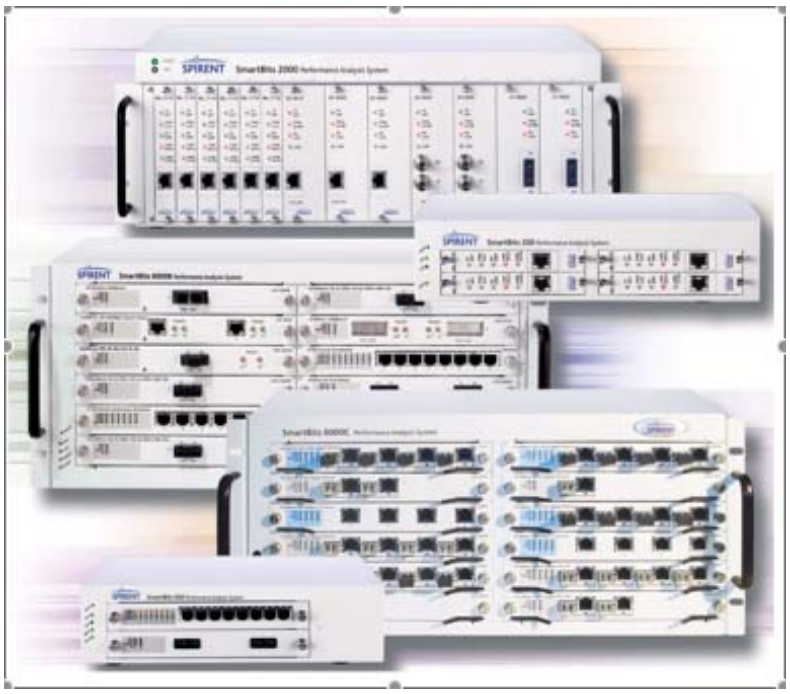

Figure 3. SmartBits chassis family.

Each chassis supports a number of SmartCards/modules, which are actually configurable network cards. Each module is connected to the network under test and is the interface for sending or receiving packets.

Any SmartsBits system supports a variety of software applications. Each application runs on PCs or workstations connected to the SmartBits chassis. Almost all SmartBits applications have an easy-to-use Graphical User Interface (GUI), together with test setup wizards and shortcut options, which greatly simplify the test setup procedure. Test results can be viewed in spreadsheet or graph form, providing complete results analysis support.

2. Smartwindow Transmit Setup

The application that provides the ability to send specially formatted packets is the SmartWindow. The process is almost simple because of the graphical interface. First, the SmartBits module must be connected to the network and the SmartBits chassis connected to the PC with the SmartWindow application installed. The GUI for SmartWindow is shown in Figure 4. This picture reflects the SmartBits chassis used. The chassis used in this case is the SmartBits 6000C, which supports two modules of which only one is needed in our case.

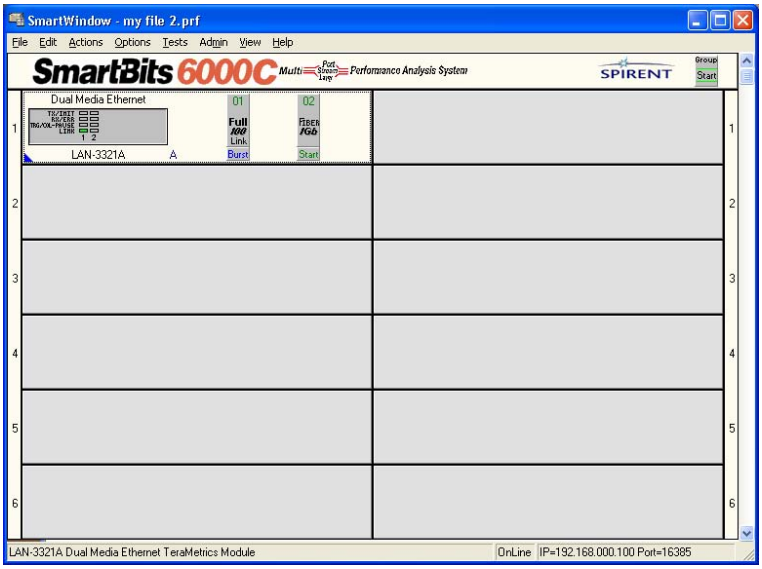

Figure 4. SmartWindow application interface.

For each module we can select the “transmit setup” wizard shown in Figure 5, and use a graphical interface to select the desired protocol to be included and set the values for most of their fields for each one of them. In Figures 6, 7, and 8 below are presented the different interfaces for the general, or basic, settings for the TCP, IPv6, and Ethernet protocol headers.

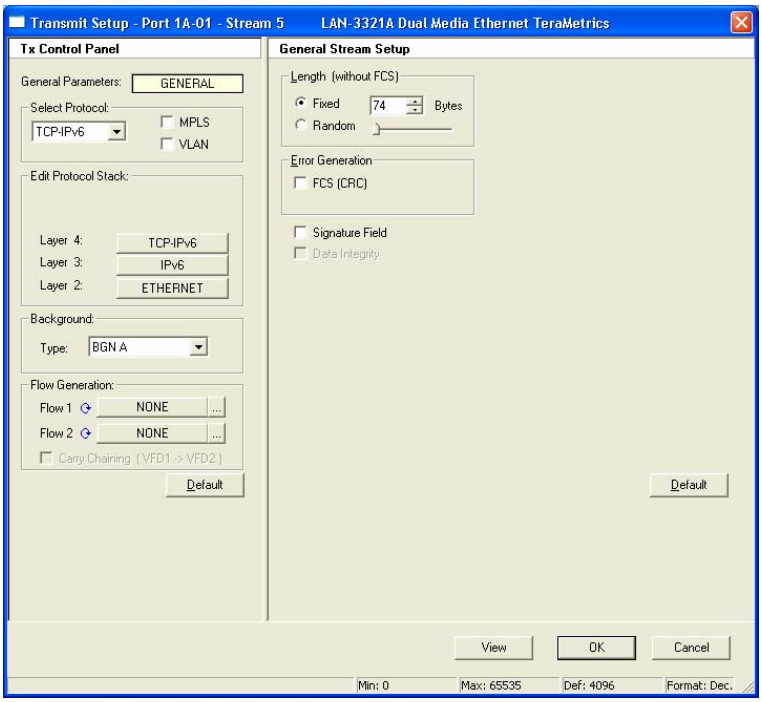

Figure 5. Transmit setup for General configuration.

In the general configuration, we set the size of the entire frame to be sent and also some predefined error generation and security features, available for use if necessary.

At the TCP header interface we can configure the following values, which are represented in the window interface with a white background:

- Source and destination port numbers
- Window size
- Initial sequence number
- Acknowledgment number
- Urgent pointer
- TCP flags set

The values that have a shaded background can not be changed. Also, in this mode we cannot set any options available at the TCP header and the two reserved flags, ECN and CWR. This is actually one restriction of this mode of operation.

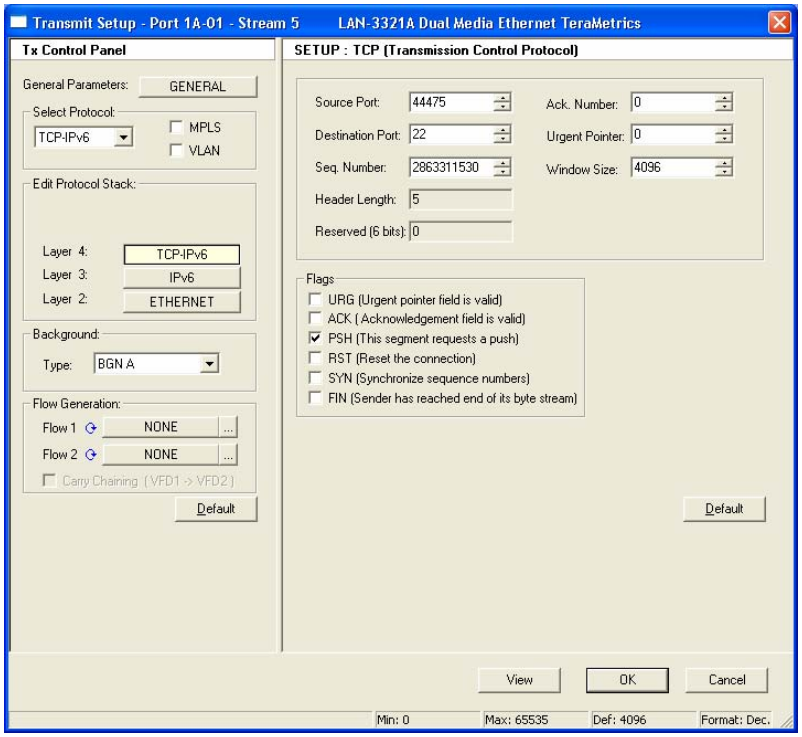

Figure 6. Transmit setup for TCP header configuration.

At the IPv6 header interface we can configure the following values, which are represented with a white background:

- Source and destination address
- Flow label
- Traffic class
- Hop limit

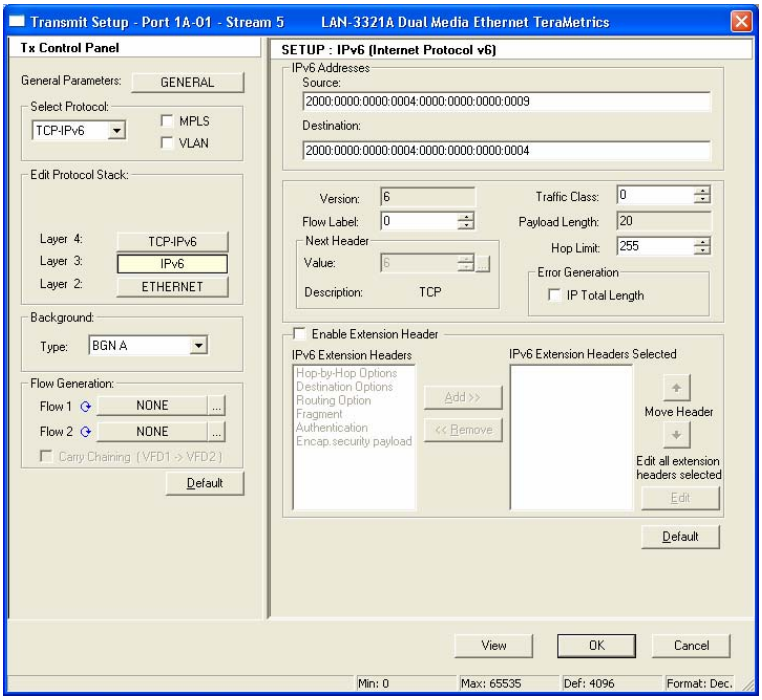

Figure 7. Transmit setup for IPv6 header configuration.

The other fields of the IPv6 header cannot be changed. Also, we can choose the "Enable Extension Header" option, in case we need to include any extension headers. These headers include the following:

- Hop-by-hop options
- Destination options
- Routing option
- Fragment

- Authentication
- Encapsulation security payload

It is possible for the user to select one or more of them to be included in the order he wishes.

At the Ethernet interface, shown in Figure 9, we can configure the source and destination MAC address. The type field of the TCP header is configured automatically by the application.

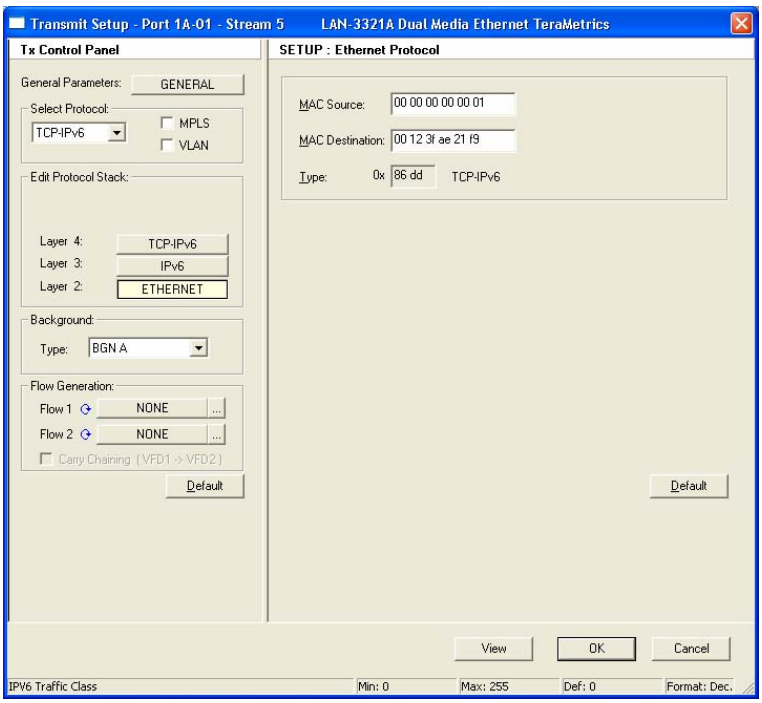

Figure 8. Transmit setup for Ethernet configuration.

Similarly, other protocol headers instead of those described earlier can be configured. Those include the UDP and IPv4 headers. Further, we could select the option to send a packet without any transport layer headers.

The other available mode of operation is one where the user constructs the entire frame to be sent. This mode is more appropriate when we need to include some TCP options in the packet. In this mode, the user must input the appropriate values for each

header in hexadecimal format and calculate the checksum value manually. Figure 9 shows the “transmit setup” wizard for this mode of operation. In this mode, the user can chose the number of packets to be sent, the length of the packet, and any application layer data to be included. More options are supported in this wizard but are irrelevant for our case.

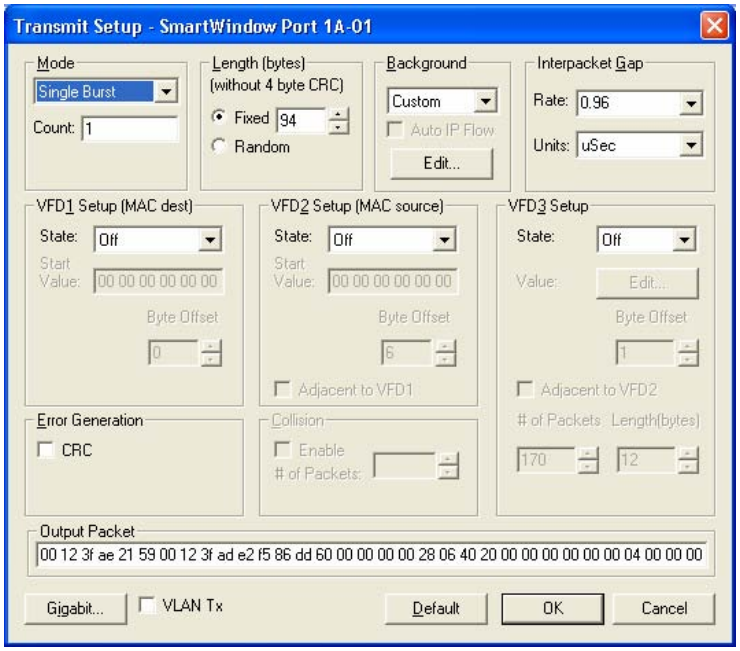

Figure 9. Transmit setup for editing the entire packet manually.

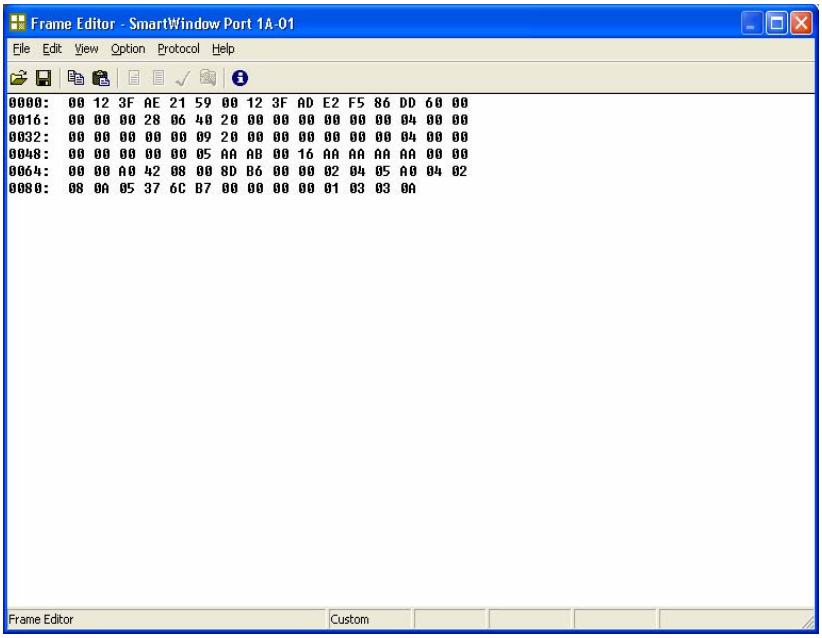

Figure 10. SmartWindow application interface.

The next step is to edit the packet, like the one presented in Figure 10. The following values, in hexadecimal, have been set for this packet.

- Destination MAC address : 00123FAE2159
- Source MAC address : 00123FADE2F5
- Type code : 86DD
- Version : 6 (IPv6 header)
- Traffic class : 00
- Flow lebel : 00000
- Payload length : 0028 (40 Bytes)
- Next header :06 (TCP)
- Hop limit :40 (64 hops)
- Source IPv6 address : 20000000000000040000000000000009
- Destination IPv6 address : 20000000000000040000000000000005
- Source port number : AAAB (decimal 43691)
- Destination port number : 0016 (decimal 22)
- Sequence number : AAAAAAAA
- Acknowledgment number : 00000000
- Header length : A (10 32-bit words)
- Reserved : 0
- Flags : 42 (ECN, SYN)
- Window size : 0800
- Checksum : 8DB6
- Urgent pointer : 0000
- Maximum segment size : 020405A0 (1440)

- SACK permitted : 0402
- Time stamp tsval , tsecr : 080A05376CB700000000
- NOP : 01
- Window scale : 03030A (10)

One difficulty we encountered with this mode of operation is the calculation of the checksum value. If the checksum is not correct, the destination will drop the packet without any notice to the source of the packet. There are some scripts available that can calculate the checksum, but in this case also the user has to provide the necessary fields included in the pseudo header and used for the checksum calculation one by one. After some experimentation, we discovered a simple work-around to find the correct value of the checksum very quickly. It has been described earlier that the OS fingerprinting process and is performed with the help of a packet analyzer. For this thesis, we used Wireshark. When a packet with an incorrect checksum value is sent through our network it will be received at the destination host and the packet analyzer. While destination will drop the packet, Wireshark will capture the packet and present the values. Wireshark will identify the wrong checksum and will make a notice that the checksum was incorrect and will include the correct value of the checksum for this packet. So, we can just send the packet once with an arbitrary value for the checksum and after we record the correct value, calculated by Wireshark, we simply apply the correction and resend the packet.

With the test bed network established and the means to generate arbitrary packets identified, we are ready to employ and assess the various means of extracting identifying characteristics of operating systems. We begin in the next chapter by looking at the results of two of these tools as applied to IPv4 and IPv6.

IV. TESTING OF EXISTING IPV4-BASED METHODS

A. INTRODUCTION

Many methods have been developed to perform OS fingerprinting in an IPv4 environment. Two of the tools that have the capability to conduct OS detection are Nmap and Queso. This chapter initially describes the methods used by these two tools and validates them against the OSes running on the machines of the network presented in Figure 2. Then follows a discussion about the most important differences between the IPv4 and the IPv6 headers and finally, the methods described earlier applied in IPv6 environment.

1. OS Detection by Nmap

Nmap was originally developed for port scanning, but later the capability for OS detection was added. Nmap, performs a 3-step procedure for OS detection. Initially it attempts to determine if the target host is “alive,” that is, if it has network connectivity. The next step is to port scan the target host. This step triggers the ports on the target and determines which of them are open or closed. Nmap needs at least one open and one closed port in order to make an accurate guess about the OS running on the target host. This process yielded the results presented in Table 3 below regarding the listening TCP ports on each OS in the network of Figure 2:

	22	111	135	139	445	1025	3689	32768	32769
MS Server 2003 Enter. Ed.			√	√	√	√			
MS Server 2003 Stand. Ed.			√	√	√	√			
Windows XP Professional			√	√	√				
Red Hat Linux Enter. 4 WS	√	√							√
Red Hat Linux 9.0	√	√						√	
Fedora core4	√	√							√
FreeBSD 6.0									
Mac OS X							√		

Table 3. Open TCP ports on each OS with IPv4 address.

The FreeBSD 6.0 had no ports open by default, thus we have to open at least one manually in order to be able to trigger it later for OS detection. Also, Nmap is not able to conduct OS fingerprinting in IPv6, but it can port scan a host by using its IPv6 address. The results from port scanning the machines using IPv6 are presented in Table 4 below.

	22	135	445	1025
MS Server 2003 Enter. Ed.		√	√	√
MS Server 2003 Stand. Ed.		√	√	√
Windows XP Professional		√		
Red Hat Linux Enter. 4 WS	√			
Red Hat Linux 9.0	√			
Fedora core4	√			
FreeBSD 6.0				
Mac OS X				

Table 4. Open TCP ports on each OS with IPv6 address.

From this table it is evident that the same services are not necessarily available with both IPv4 and IPv6, at least by default. In this case also, FreeBSD 6.0 and the Mac OS X didn't have any open ports, thus we need to open at least one manually for both.

Finally, after the open ports have been found, Nmap sends a series of specially formatted TCP and UDP packets and then examines the responses from the target machine. Nmap examines the fields in the headers of the responses and compares them against a database of known fingerprints[4]. If there is a match it will come up with an inference about the OS running on the target machine. More information about the capabilities of Nmap can be found at www.insecure.org/nmap/data/nmap_manpage.html.

2. OS Detection By Queso

Queso is another tool for OS detection in an IPv4 environment. However, it is much simpler than Nmap. This program makes the assumption that we already know at least one open port on the target machine, so we have just to specify the IP address and the open port. Queso uses seven methods to detect the OS. That means seven packets,

[4] There are about 1,500 fingerprints in the database

formatted appropriately, are sent to the target. Then some of the values in the headers of the responses are examined against a list of known fingerprints, as is the case with Nmap, and if there is a match, it will come up with a guess about the OS running on the target.

B. VALIDATION IN IPV4 ENVIRONMENT

Now that we have seen the general view of how those tools perform the OS fingerprinting we can test them against the machines of our network and develop a solid baseline for OS fingerprinting in IPv4.

1. Test Cases by Nmap

The methods utilized by Nmap are described below and the results for each one of them gathered for each OS are presented in the following Tables.

a. Test case 1, SYN, ECN packet with options to an open port

This method sends a packet with the SYN and ECN bits set and some options included in the TCP header. This packet is a request for setting up a connection with the remote host on the port number specified in the port number field of the packet send. The common behavior of the host, which has the port listening for incoming connections, is to acknowledge the request and respond with a packet with the SYN and ACK flag bits set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 5.

b. Test case 2, NULL packet with options to an open port

This method sends a packet with no flags set and some options included in the TCP header. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 6.

c. Test case 3, FIN, SYN, PSH, URG packet with options to an open port

This method sends a packet with the FIN, SYN, PSH, and URG bits set and some options included in the TCP header. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 7.

d. Test case 4, ACK packet with options to an open port

This method sends a packet with the ACK flag set and some options included in the TCP header. This packet originally acknowledges a packet sent from the

remote host. Since no packet was sent earlier from that host, the common response is for the target host to send back a packet with the reset (RST) flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 8.

e. Test case 5, SYN packet with options to a closed port

This method sends a packet with the SYN flag set and some options included in the TCP header to a closed port. This is a request to establish a connection with the remote host on the specified port number. Because the port is closed, the remote host must reject the request and send back a packet with the RST and ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 9.

f. Test case 6, ACK packet with options to a closed port

This method sends a packet with the ACK flag set and some options included in the TCP header to a closed port. The ACK flag indicates that the sender acknowledges some data sent from the remote host. In this case, however, the remote host has neither sent any data nor is even listening to the port for incoming packets. Thus, the remote host sends back a packet with the RST flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 10.

g. Test case 7, FIN, PSH, URG packet with options to a closed port

This method sends a packet with the FIN, PSH, and URG flags set and some options included in the TCP header to a closed port. The FIN flag in the header indicates that the sender is attempting to close a connection, but because there is not any active connection, the remote host sends back a packet with the RST and ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 11.

h. Test case 8, UDP packet with data to a closed port

This method sends a UDP packet to a closed port. The common response is for the remote host to send back an ICMP port unreachable packet (type 3, code 3). A summary of the packet sent to each host in the network and the responses received from them is presented in Table 12.

Test Case 1-Nmap

SYN, ECN packet with options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ[5]
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	SYN, ECN
	OPTIONS	window scale : 10 NOP max seg size : 265 time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	60	60	60	60	60	60	60	60
	FLAGS	0	0	DF	DF	DF	DF	DF	DF
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN[6]	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++[7]	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	HEADER	40	40	40	40	40	40	40	40
	WINDOW SIZE	16384	16384	16430/65535[8]	5792	5792	5792	65535	65535
	OPTIONS	max seg size : 1460 NOP window scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP window scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP window scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP NOP Time stamp : X, Y[9] NOP window scale : 2(x4)	max seg size : 1460 NOP NOP Time stamp : X, Y NOP window scale : 2(x4)	max seg size : 1460 NOP NOP Time stamp : X, Y NOP window scale : 0(x1)	max seg size : 1460 NOP window scale : 1(x2) NOP NOP Time stamp X, Y	max seg size : 1460 NOP window scale : 0(x1) NOP NOP Time stamp X, Y

Table 5. Test case 1 by Nmap, SYN, ECN packet with options to an open port

[5] A randomly selected Initial Sequence number

[6] The value is selected from the OS

[7] The next expected sequence number

[8] It has been observed that Windows XP Pro may use either values for their Window Size

[9] X, Y are tsval and tsecr values that change over time and set by the OS

Test Case 2- Nmap

NULL packet with options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	0
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20					
	TOS	0	0	0					
	TOTAL LENGTH	40	40	40					
	FLAGS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	TTL	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ	SEQ	SEQ					
	HEADER	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					
	OPTIONS	--	--	--					

Table 6. Test case 2 by Nmap, NULL packet with options to an open port

Test Case 3- Nmap

FIN, SYN, PSH, URG packet with options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	FIN, SYN, PSH, URG
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH								
	TOS	0	0	0	0	0	0	0	
	TOTAL LENGTH	40	40	40	40	40	40	40	NO REPLY
	FLAGS	0	0	DF	DF	DF	DF	DF	
	TTL	128	128	128	64	64	64	64	
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	
	HEADER	40	40	40	40	40	40	40	
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	
	WINDOW SIZE	16384	16384	16430/65535	5792	5792	5792	65535	
	OPTIONS	max seg size : 1460 NOP Win scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP Win scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP Win scale : 0(x1) NOP NOP Time stamp : 0, 0	max seg size : 1460 NOP NOP Time stamp : X, Y NOP Win scale : 2(x4)	max seg size : 1460 NOP NOP Time stamp : X, Y NOP Win scale : 2(x4)	max seg size : 1460 NOP NOP Time stamp : X, Y NOP Win scale : 0(x1)	max seg size : 1460 NOP Win scale : 1(x2) NOP NOP Time stamp : X, Y	

Table 7. Test case 3 by Nmap, FIN, SYN, PSH, URG packet with options to an open port

Test Case 4- Nmap

ACK packet with options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	ACK
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	0
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	

Table 8. Test case 4 by Nmap, ACK packet with options to an open port

Test Case 5- Nmap

SYN packet with options to a closed port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	SYN
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	0
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	

Table 9. Test case 5 by Nmap, SYN packet with options to a closed port

Test Case 6- Nmap

ACK packet with options to a closed port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	ACK
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	0
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	

Table 10. Test case 6 by Nmap, ACK packet with options to a closed port

Test Case 7- Nmap

FIN, PSH, URG packet to a closed port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	60
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	FIN, PSH, URG
	OPTIONS	window scale : 10 NOP max seg size : 265 Time stamp : X, 0 EOL

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	0
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ	SEQ
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	

Table 11. Test case 7 by Nmap, FIN, PSH, URG packet with options to a closed port

Test Case 8- Nmap

UDP packet with data to a closed port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	328
	FLAGS	0
UDP	DATA	300 Bytes
	LENGTH	308

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	1100 00 0 0	1100 00 0 0	1100 00 0 0	0	0
	TOTAL LENGTH	176	176	176	356	356	356	56	56
	FLAGS	0	0	0	0	0	0	0	0
	TTL	128	128	128	64	64	64	64	64
ICMP	TYPE	3	3	3	3	3	3	3	3
	CODE	3	3	3	3	3	3	3	3
	DATA	120 Bytes	120 Bytes	120 Bytes	300 Bytes	300 Bytes	300 Bytes	NO DATA	NO DATA

Table 12. Test case 8 by Nmap, UDP packet with data to a closed port

2. Test Cases By Queso

The methods implemented by Queso are described below and the results from using Queso against the OSes running in the network of Figure 2 are presented in the following Tables.

a. Test case 1, SYN packet without options to an open port

In this method, a common request for setting up a connection[10] with the target host on the specified port number is sent to that machine. The common action for a machine that listens for incoming requests is to accept the request and respond with a SYN/ACK packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 13.

b. Test case 2, SYN, ACK packet without options to an open port

In this method, a packet is sent that actually accepts an incoming request to set up a connection. Because no request has been sent from the remote machine, that machine will send back a RST packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 14.

c. Test case 3, FIN packet without options to an open port

In this method, a packet is sent with the FIN flag set. In this case, the way the remote machines respond may vary. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 15.

d. Test case 4, FIN, ACK packet without options to an open port

In this method, a packet is sent with the FIN and ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 16.

e. Test case 5, SYN, FIN packet without options to an open port

In this method, a packet is sent with the SYN and FIN flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 17.

[10] This is the first part of the 3-way handshake between client and server

f. Test case 6, PSH packet without options to an open port

In this method, a packet is sent with the PSH flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 18.

g. Test case 7, SYN, ECN, CWR packet without options to an open port

In this method, a packet is sent with the SYN flag and two more flags that are not used in a TCP/IP network, like ECN and CWR, set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 19.

Test Case 1-Queso

SYN packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	44	44	44	44	44	44	44	44
	FLAGS	0	0	DF	DF	DF	DF	DF	DF
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	24	24	24	24	24	24	24	24
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	WINDOW SIZE	16384	16384	16616/65535	5840	5840	5840	65535	65535
	OPTIONS	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460

Table 13. Test case 1 by Queso, SYN packet without options to an open port

Test Case 2-Queso

SYN, ACK packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, ACK

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	0
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0

Table 14. Test case 2 by Queso, SYN, ACK packet without options to an open port

Test Case 3-Queso

FIN packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	FIN

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	TOS	0	0	0					
	TOTAL LENGTH	40	40	40					
	FLAGS	0	0	0					
	TTL	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++					
	HEADER	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					

Table 15. Test case 3 by Queso, FIN packet without options to an open port.

Test Case 4-Queso

FIN, ACK packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	FIN, ACK

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	40	40	40	40	40	40	40	40
	FLAGS	0	0	0	DF	DF	DF	DF	DF
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0

Table 16. Test case 4 by Queso, FIN, ACK packet without options to an open port.

Test Case 5-Queso

SYN, FIN packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, FIN

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	
	TOS	0	0	0	0	0	0	0	
	TOTAL LENGTH	44	44	44	44	44	44	44	
	FLAGS	0	0	DF	DF	DF	DF	DF	NO REPLY
	TTL	128	128	128	64	64	64	64	
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	
	HEADER	24	24	24	24	24	24	24	
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	
	WINDOW SIZE	16384	16384	65535	5840	5840	5840	65535	
	OPTIONS	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	

Table 17. Test case 5 by Queso, SYN, FIN packet without options to an open port

Test Case 6-Queso

PSH packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	PSH

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20					
	TOS	0	0	0					
	TOTAL LENGTH	40	44	44					
	FLAGS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	TTL	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ	SEQ	SEQ					
	HEADER	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					

Table 18. Test case 6 by Queso, PSH packet without options to an open port

Test Case 7-Queso

SYN, ECN, CWR packet without options to an open port

Packet send

IPV4	HEADER LENGTH	20
	TOS	0
	TOTAL LENGTH	40
	FLAGS	0
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, ECN, CWR

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP 2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV4	HEADER LENGTH	20	20	20	20	20	20	20	20
	TOS	0	0	0	0	0	0	0	0
	TOTAL LENGTH	44	44	44	44	44	44	44	44
	FLAGS	0	0	DF	DF	DF	DF	DF	DF
	TTL	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	24	24	24	24	24	24	24	24
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	WINDOW SIZE	16384	16384	65535	5840	5840	5840	65535	65535
	OPTIONS	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460	max seg size : 1460

Table 19. Test case 7 by Queso, SYN, ECN, CWR packet without options to an open port

3. Analysis

From the results presented in the tables above, it is evident that not all OSes respond with the same values in their headers. Further, there are cases where they don't even exhibit the same behavior. The key differences are highlighted below.

a. Don't Fragment Bit (DF)

Not all OSes set the don't fragment bit in the IPv4 header. The Windows servers never set the don't fragment bit in their response, and Windows XP sets this flag only when it responds with a SYN, ACK packet. The other OSes always set the DF bit except in the case of the ICMP response.

b. TTL Value

The Windows OSes always set this value to 128 and the other OSes always use the value 64. The TTL value, however, cannot be used explicitly to characterize the OS. This is because the TTL is set from the sender of the packet to some initial value and decreases by one every time the packet visits an intermediate node. Thus, this value can only be used implicitly to exclude OSes. For example, if the TTL value in a received packet is between 64 and 128, that host could not be an OS that sets the initial TTL value to 64 or possibly less. However the TTL can be configured to an other default value and lead to inaccurate conclusions. Thus it can not be considered as an accurate factor for detecting an OS.

c. Window Size

The window size is actually the buffer size in bytes set at the remote host for this connection. This value is an important factor for OS detection because many OSes have a unique value among the other OSes that by itself could possibly identify the OS. Also, it was observed that the Windows and the Linux machines change their window size value depending on whether the packet sent to that host included options. In contrast, the FreeBSD machine always set this value to maximum (65535 bytes). So, the Test Cases 1 and 3 with Nmap included options in the TCP headers of the packets sent to the target host. The responses received from the Windows servers had the window size value set to 16384, Windows XP Pro set the value to either 16430 or 65535, the Linux machines set that value to 5792, and the FreeBSD set the value to 65535. In the Test Cases 1, 5, and 7 with Queso, there were no options included in the packet sent and the

responses received from the Windows servers had the window size value set to 16384 again, Windows XP Pro set the value to 16616 or 65535, the Linux machines set it to 5840, and the FreeBSD set the value to 65535.

d. Options

The options field is another key point by which to differentiate OSes. Not all OSes support all options. For those options supported, they do not list the supported options in the same order. Furthermore, it was observed that, while some OSes may support the same options and present them in the same order, they may support different values for the window scale, This is the case with Fedora Core 4 and Red Hat Linux 9.0, where Fedora Core 4 sets that value to x4 and Red Hat Linux 9.0 sets the value to x1.

e. Initial Sequence Number

The initial sequence number is the value chosen by the OS to start counting the bytes in the packets it sends to the other party of the connection. Although the RFC 793 does not specify any fixed pattern for choosing the ISN, the selection must ensure that no other packet belonging to the same connection has the same sequence number. In the tests presented above, the value of the sequence number is presented with SEQ, referring to some randomly generated value. However, Nmap records the ISNs selected from the remote machine and attempts to find a pattern in the selection process that the OS developer may have used.

f. No Reply

There are some cases where the RFCs do not specify clearly what the appropriate response should be from a host receiving some types of packets. Thus, the various developers of the TCP/IP stacks implement different behaviors for their OSes. Examples of this are the test cases of the NULL packet, PSH packet, and the FIN packet as presented in the Tables 2, 11, and 14 respectively. In those cases, only the Windows machines sent back a response.

g. ICMP Port Unreachable

Finally, another factor that helps in the OS detection is the amount of original data some OSes include in the “ICMP port unreachable” message they send back. The purpose of this inclusion is for the other host to be able to identify which

packet initiated the ICMP message. However, not all OSes include the same amount of data, as is apparent in the case of a UDP packet sent to a closed port shown in Table 8.

C. APPLICABILITY TO IPV6 ENVIRONMENT

The previous section discussed OS fingerprinting as it may be conducted in an IPv4 environment using two different tools available today, Nmap and Queso. These tools use the same concept regarding OS detection. Both attempt to detect the OS by triggering the target host to respond to specially constructed packets and then comparing the responses to a database containing known fingerprints. If a match is found, both tools will make a guess about the OS running on the target host. However, these tools use different methods for triggering the protocol stack of the target host. Each method sends a different kind of packet in order to trigger a specific response, which hopefully will include unique values for some fields among the OSes. That means that the decision as to the kind of the packet to be sent is crucial for the development of the database containing the fingerprints in the first place.

This section describes an attempt to apply the same methods used by Nmap and Queso in an IPv6 environment and examines the existence of identifying factors among the OSes of the network of Figure 2. In order to proceed to the actual tests on IPv6 it is important to have a clear knowledge of the IPv6 header format. Also, because the objective of this section is to import methods from the IPv4 protocol, we should identify the differences and similarities between IPv4 and IPv6 headers. A discussion of these aspects is presented in the following two paragraphs.

1. IPv6 vs IPv4 Header Format

Any packets sent across the IPv6 network must comply with the IPv6 protocol and, therefore, must use the format and syntax defined by the RFC 2460. Thus, it is evident that the format of the packets as they were constructed and sent by the tools over the IPv4 protocol cannot be sent exactly the same way over the IPv6 protocol. This is because IPv4 and IPv6 protocols are specified by different RFCs and use different formats for their headers.

The format of the IPv4 and IPv6 headers are presented in the Figures 11 and 12 below. The most important differences between these two header formats are the following:

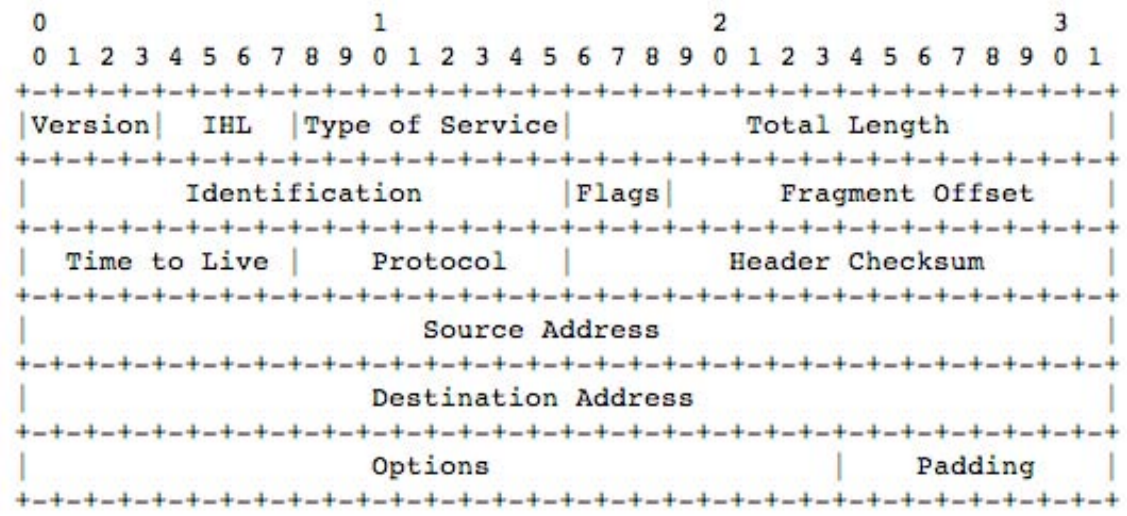

Figure 11. IPv4 header format.

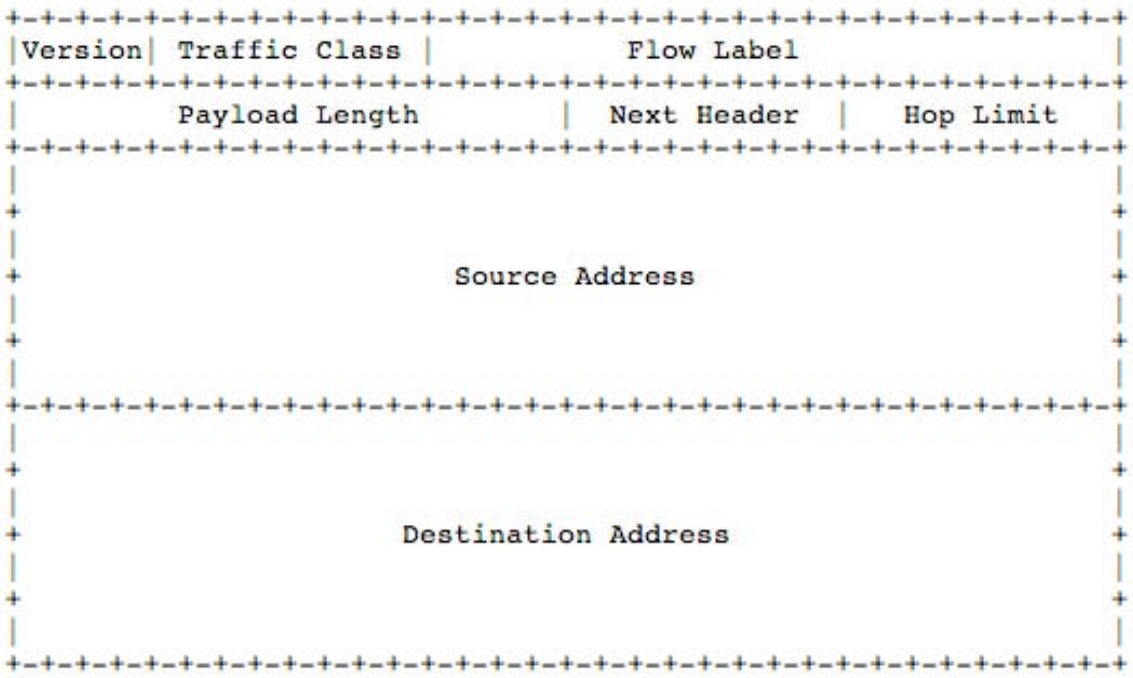

Figure 12. IPv6 header format.

a. Header Length

The header length in the IPv6 header is exactly 40 bytes long, instead of the variable length in the IPv4 header. This is the reason why there is no header length field in the IPv6 header.

The variable length in the IPv4 header is the result of the options field at the end of the header. As not all hosts support all available options, and because not all options are needed for every packet sent across the network, they are not necessarily included in the header, so the IPv4 header has a variable length. However, in the IPv6 header this field has been removed, so that the header has a constant length.

b. Traffic Class

The traffic class field is 8 bits long and is similar to the type of service (TOS) field of the of the IPv4 header. The traffic class field in the IPv6 header is available for use by originating hosts and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets.

c. Flow Label

The flow label field is 20 bits long and, as it is described in RFC 2460, is still experimental. There is no equivalent field in the IPv4 header. Its purpose is for a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service.

d. Payload Length

This field is 16 bits long and it is similar to the total length field of the IPv4 header. The difference is that the payload length gives the number of the bytes following the IPv6 header. That means that any extension headers are included in the Payload Length.

e. Next Header

This field is 8 bits long and it is equivalent to the protocol field in IPv4 header. This field identifies the protocol to which the data of the datagram will be delivered and it uses the same values as IPv4, as described in RFC 1700.

f. Hop Limit

The hop limit field is 8 bits long and is similar with the time to live (TTL) field in the IPv4 header. The value of the hop limit is initialized by the originator of the packet and it is decremented by one by each node that forwards the packet.

g. Source and Destination Address

These two values are 128bits long each and are analogous to the source and destination address fields in the IPv4 header.

h. Fragmentation

The IPv6 protocol does not support fragmentation and reassembly at the intermediate nodes. Those operations are only performed at the source and destination hosts. That is why there are no Identification, Flags, and Fragment Offset fields included in the IPv6 header. However, if a packet is received that is too large to be forwarded to the outgoing link, the router will drop that packet and send to the packet originator an ICMP error message “Packet Too Big”.

i. Header Checksum

The header checksum field that was included in the IPv4 header has been removed from the IPv6 header. This is because the TCP and UDP protocols at the transport layer and the data links protocols, like Ethernet, also include a checksum in their headers. It seems that this field was redundant and so it was removed.

j. Options

The options field that was available in the IPv4 header has been removed from IPv6. However, it has not been removed from the protocol stack completely. That is, it has changed so that the available options become a separate header and are pointed to by the Next Header field in the IPv6 header. So, if there is a need for a specific kind of option or options to be included in the packet sent, a separate header for each of them will be added between the IPv6 and the transport layer header.

2. Equivalence Of IPv6 To IPv4 Header Fields

The methods described earlier in this chapter for OS detection over IPv4 send complete frames by generating headers of the other layers of the protocol stack. That is, they should include a transport layer header, a network layer header, and a data link layer

header. The transport and data link layer headers are not affected by the change at the network layer due to IPv6 protocol entry. This is a characteristic of the layered architecture of the protocol stack. It is possible to use the same format for these two layers over both IPv4 and IPv6 protocols. Thus, there is no doubt about the applicability of the methods that use values of the Transport layer headers for identifying the different OSes. One point of which we should be aware is that the maximum segment size that is used in the options of TCP header is 1460 bytes for IPv4 packets but in the case of IPv6 this value should be 1440 bytes. That is because the IPv6 header is 20 bytes longer than the smallest IPv4 header. It should not be expected that the OSes will use the same values in both cases. For example, it may be possible to identify the OS based on the window size value used in the TCP header, but this doesn't necessarily means that this value will be the same over the IPv4 and IPv6 protocol.

It should be observed that although the transport and IP layers are separate, the vendors of the OSes develop their protocol stack with these two layers very tightly coupled. Also, the OS should be able to implement both IPv4 and IPv6, thus they develop their dual protocol stack in a way that could be depicted graphically in Figure 13.

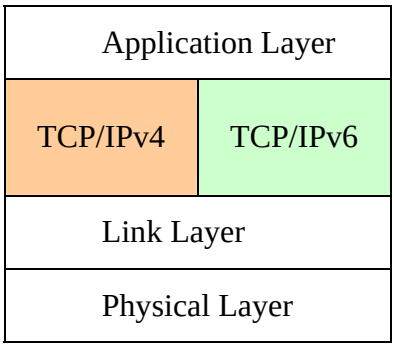

Figure 13. Dual protocol stack.

Although the transport and data link headers can be used exactly the same when probing the IPv6 protocol, that is not the case for IP header itself. Here some modifications should be applied in order to send packets over IPv6. It was noted earlier that not all fields of IPv4 have an equivalent in IPv6. Also, some fields have been

removed and some others have been added in the IPv6 header. The appropriate modifications to the packets sent over IPv4 are the following:

- Version: It must change from 4 to 6.
- Header length: There is no equivalent in IPv6 and it can be ignored
- Type of service: It is equivalent with the traffic class in IPv6. No default values should be tested.
- Total length: There is no equivalent in IPv6 and it should be ignored.
- Identification/flags/fragment offset: These values do not have an equivalent in the IPv6 and should be ignored. However, the DF flag was a key factor for identifying among OSes. So, this identifying factor will not be available in IPv6.
- Time to live: It is equivalent to hop limit in the IPv6, so it can have the same or different values.
- Protocol: This is equivalent to the next header in the IPv6. Both protocols use the values described in RFC 1700.
- Internet checksum: There is no equivalent field in the IPv6 header and it can be ignored.
- Source/destination address: This is analogous to the source and destination address in the IPv6. However, in the IPv6 header these addresses are 128 bits long. They are still used for identifying the source and destination.
- Options: There is no equivalent field in the IPv6 header. Instead, a separate header is added between the IPv6 and transport layer headers for any desired option, which is identified through the next header field in the IPv6 header.

D. APPLICABILITY OF KNOWN METHODS OVER IPV6 PROTOCOL

Eight methods used from Nmap and seven more from Queso were explored earlier in this chapter. We turn now to the applicability of those methods for distinguishing the OS when employing the IPv6 protocol.

1. Applicability of the Methods Used by Nmap

a. Test case 1, SYN, ECN packet with options to an open port

This method sends a packet with the SYN and ECN bits set and some options included in the TCP header. This packet is a request for setting up a connection with the remote host on the port number specified in the port number field of the packet sent. The common behavior of the host which has the port listening for incoming connections is to acknowledge the request and respond with a SYN, ACK packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 20.

b. Test case 2, NULL packet with options to an open port

This method sends a packet with no flags set and some options included in the TCP header. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 21.

c. Test case 3, FIN, SYN, PSH, URG packet with options to an open port

This method sends a packet with the FIN, SYN, PSH, URG bits set and some options included in the TCP header. A summary of the packet sent to each host in the network and the responses received from each them is presented in Table 22.

d. Test case 4, ACK packet with options to an open port

This method sends a packet with the ACK flag set and some options included in the TCP header. This packet originally acknowledges a packet sent from the remote host. Because no packet was sent from that host, the common response is for the target host to send back a packet with the RST flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 23.

e. Test case 5, SYN packet with options to a closed port

This method sends a packet with the SYN flag set and some options included in the TCP header to a closed port. This is a request to establish a connection with the remote host on the specified port number. Because the port is closed, the remote

host must reject the request and send back a packet with the RST and ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 24.

f. Test case 6, ACK packet with options to a closed port

This method sends a packet with the ACK flag set and some options included in the TCP header to a closed port. The ACK flag indicates that the sender acknowledges some data sent from the remote host. In this case, however, the remote host has neither sent any data nor is it listening to the indicated port for incoming packets. Thus, the remote host sends back a packet with the RST flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 25.

g. Test case 7, ACK packet with options to a closed port

This method sends a packet with the FIN, PSH, and URG flags set and some options included in the TCP header to a closed port. The FIN flag in the header indicates that the sender is attempting to close a connection, but because there is no active connection, the remote host sends back a packet with the RST/ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 26.

h. Test case 8, UDP packet with data to a closed port

This method sends a UDP packet to a closed port. The expected response is for the remote host to send back an ICMP error message, “port unreachable” (type 1, code 4). A summary of the packet sent to each host in the network and the responses received from them is presented in Table 27.

Test Case 1-Modified from Nmap

SYN, ECN packet with options to a open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	SYN, ECN
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	00/03/08/0D/0E	0
	FLOW LABEL	0	0	0	0	0	0	RANDOM#	RANDOM#
	PAYLOAD	24	24	24	40	40	40	44	40
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	24	24	24	40	40	40	44	40
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	WINDOW SIZE	17280	17280	17280	5712	5712	5712	65535	65535
	OPTIONS	Max seg size : 1440	Max seg size : 1440	Max seg size : 1440	Max seg size : 1440 SACK permitted Time Stamp : X, NOP Win Scale : 2(X4)	Max seg size : 1440 SACK permitted Time Stamp : X, Y NOP Win Scale : 2(X4)	Max seg size : 1440 SACK permitted Time Stamp : X, Y NOP Win Scale : 0(X1)	Max seg size : 1440 NOP Win Scale : 1(X2) NOP NOP Time Stamp : X, Y EOL	Max seg size : 1440 NOP Win Scale : 0(X1) NOP NOP Time Stamp : X, Y

Table 20. Test case 1 modified from Nmap, SYN, ECN packet with options to an open port.

Test Case 2- Modified from Nmap

NULL packet with options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	FLAGS	--
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	FLOW LABEL	0	0	0					
	PAYLOAD	20	20	20					
	HOP LIMIT	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ	SEQ	SEQ					
	HEADER LENGTH	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					
	OPTIONS	--	--	--					

Table 21. Test case 2 modified from Nmap, NULL packet with options to an open port.

Test Case 3- Modified from Nmap

SYN, FIN, PSH, URG packet with options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	FLAGS	SYN, FIN, PSH, URG
	HEADER	40
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 EnterpriseEnterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	NO REPLY
	FLOW LABEL	0	0	0	0	0	0	RANDOM	
	PAYLOAD	24	24	24	40	40	40	44	
	HOP LIMIT	128	128	128	64	64	64	64	
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	
	HEADER LENGTH	24	24	24	40	40	40	44	
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	
	WINDOW SIZE	17280	17280	17280	5712	5712	5712	65535	
	OPTIONS	Max seg size : 1440	Max seg size : 1440	Max seg size : 1440	Max seg size : 1440 SACK permitted Time Stamp : X, Y NOP Win Scale : 2(X4)	Max seg size : 1440 SACK permitted Time Stamp : X, Y NOP Win Scale : 2(X4)	Max seg size : 1440 SACK permitted Time Stamp : X, Y NOP Win Scale : 0(X1)	Max seg size : 1440 NOP Win Scale : 1(X2) NOP NOP Time Stamp : X, Y SACK permitted EOL	

Table 22. Test case 3 modified from Nmap, FIN, SYN, PSH, URG packet with options to an open port

Test Case 4- Modified from Nmap

ACK packet with options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	40
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER LENGTH	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 23. Test case 4 modified from Nmap, ACK packet with options to an open port

Test Case 5- Modified from Nmap

SYN packet with options to a closed port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	FLAGS	SYN
	HEADER	40
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 2(x4)

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 24. Test case 5 modified from Nmap, SYN packet with options to a closed port.

Test Case 6- Modified from Nmap

ACK packet with options to a closed port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER LENGTH	40
	FLAGS	ACK
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 25. Test case 6 modified from Nmap, ACK packet with options to a closed port.

Test Case 7- Modified from Nmap

FIN, PSH, URG packet with options to a closed port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	40
	HOP LIMIT	64
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER LENGTH	40
	FLAGS	FIN, PSH, URG
	OPTIONS	max seg size : 1440 SACK permitted Time stamp : X, 0 NOP Win Scale : 10

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ	SEQ
	HEADER LENGTH	20	20	20	20	20	20	20	20
	FLAGS	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK	RST, ACK
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 26. Test case 7 modified from Nmap, FIN, PSH, URG packet with options to a closed port.

Test Case 8- Modified from Nmap

UDP packet to a closed port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	1208
	HOP LIMIT	64
UDP	DATA	1200 Bytes

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	0	0
	PAYLOAD	1240	1240	1240	1240	1240	1240	1240	1240
	HOP LIMIT	128	128	128	64	64	64	64	64
ICMP	TYPE	1	1	1	1	1	1	1	1
	CODE	4	4	4	4	4	4	4	4
	DATA	1184 Bytes	1184 Bytes	1184 Bytes	1184 Bytes	1184 Bytes	1184 Bytes	1184 Bytes	1184 Bytes

Table 27. Test case 8 modified from Nmap, UDP packet with data to a closed port.

2. Applicability of the Methods Used by Queso

a. Test case 1, SYN packet without options to an open port

In this method, a common request for setting up a connection[11] with the target host, on a specified port number, is sent to the target host. The expected action for a machine which listens for incoming requests is to accept the request and respond with a SYN, ACK packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 28.

b. Test case 2, SYN, ACK packet without options to an open port

In this method, a packet is sent that actually accepts an incoming request to set up a connection. Because no request has been sent from the target host, that host will send back a RST packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 29.

c. Test case 3, FIN packet without options to an open port

In this method, a packet is sent with the FIN flag set. In this case, the way the remote machines respond may vary. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 30.

d. Test case 4, FIN, ACK packet without options to an open port

In this method, a packet is sent with the FIN and ACK flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 31.

e. Test case 5, SYN, FIN packet without options to an open port

In this method, a packet is sent with the SYN and FIN flags set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 32.

f. Test case 6, PSH packet without options to an open port

In this method, a packet is sent with the PSH flag set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 33.

[11] This is the first part of the 3-way handshake between client and server.

g. Test case 7, SYN, ECN, CWR packet without options to an open port

In this method, a packet is sent with the SYN flag and two additional flags that are not used in a TCP/IP network, such as ECN and CWR, set. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 34.

Test Case 1- Modified from Queso

SYN packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN

Packet received

		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	RANDOM	RANDOM
	PAYLOAD	24	24	24	24	24	24	24	24
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	24	24	24	24	24	24	24	24
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	WINDOW SIZE	17080	17080	17080	5760	5760	5760	65535	65535
	OPTIONS	max seg size: 1440	max seg size: 1440	max seg size: 1440	max seg size: 1440	max seg size: 1440	max seg size: 1440	max seg size: 1440	max seg size: 1440

Table 28. Test case 1 modified from Queso, SYN packet without options to an open port

Test Case 2- Modified from Queso

SYN, ACK packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, ACK

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 29. Test case 2 modified from Queso, SYN, ACK packet without options to an open port.

Test Case 3- Modified from Queso

FIN packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	FIN

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	FLOW LABEL	0	0	0					
	PAYLOAD	20	20	20					
	HOP LIMIT	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++					
	HEADER	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					
	OPTIONS	--	--	--					

Table 30. Test case 3 modified from Queso, FIN packet without options to an open port

Test Case 4- Modified from Queso

FIN, ACK packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	FIN, ACK

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	X
	FLOW LABEL	0	0	0	0	0	0	0	Y
	PAYLOAD	20	20	20	20	20	20	20	20
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	0	0	0	0	0	0	0	0
	ACKNOWLEDGE	0	0	0	0	0	0	0	0
	HEADER	20	20	20	20	20	20	20	20
	FLAGS	RST	RST	RST	RST	RST	RST	RST	RST
	WINDOW SIZE	0	0	0	0	0	0	0	0
	OPTIONS	--	--	--	--	--	--	--	--

Table 31. Test case 4 modified from Queso, FIN, ACK packet without options to an open port

Test Case 5- Modified from Queso

SYN, FIN packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, FIN

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	NO REPLY
	FLOW LABEL	0	0	0	0	0	0	RANDOM	
	PAYLOAD	24	24	24	24	24	24	24	
	HOP LIMIT	128	128	128	64	64	64	64	
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	
	HEADER	24	24	24	24	24	24	24	
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	
	WINDOW SIZE	17080	17080	17080	5760	5760	5760	65535	
	OPTIONS	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	

Table 32. Test case 5 modified from Queso, SYN, FIN packet without options to an open port

Test Case 6- Modified from Queso

PSH packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	PSH

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	FLOW LABEL	0	0	0					
	PAYLOAD	20	20	20					
	HOP LIMIT	128	128	128					
TCP	SEQUENCE	0	0	0					
	ACKNOWLEDGE	SEQ	SEQ	SEQ					
	HEADER	20	20	20					
	FLAGS	RST, ACK	RST, ACK	RST, ACK					
	WINDOW SIZE	0	0	0					
	OPTIONS	--	--	--					

Table 33. Test case 6 modified from Queso, PSH packet without options to an open port

Test Case 7- Modified from Queso

SYN, ECN, CWR packet without options to an open port

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	20
	HOP LIMIT	255
TCP	SEQUENCE	SEQ
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN, ECN, CWR

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	RANDOM	RANDOM
	PAYLOAD	24	24	24	24	24	24	24	24
	HOP LIMIT	128	128	128	64	64	64	64	64
TCP	SEQUENCE	ISN	ISN	ISN	ISN	ISN	ISN	ISN	ISN
	ACKNOWLEDGE	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++	SEQ++
	HEADER	24	24	24	24	24	24	24	24
	FLAGS	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK	SYN, ACK
	WINDOW SIZE	17080	17080	17080	5760	5760	5760	65535	65535
	OPTIONS	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440	max seg size:1440

Table 34. Test case 7 modified from Queso, SYN, ECN, CWR packet without options to an open port

3. Analysis

From the results presented in the tables above, it is evident that not all OSes respond with the same values in their headers and there are cases where they don't even exhibit the same behavior. These results parallel those found when the two tools are used in an IPv4 environment. The key points where differences can be identified between the responses or behavior of individual OSes are discussed below.

a. Window Size

The widow size is actually the buffer size in bytes, set at the remote host for this connection. This value is an important factor of OS detection because many OS use a unique value as compared to the other OSes. Also, it was observed that the Windows and the Linux machines change their window size value depending on whether or not the packet sent to that host included options, while the FreeBSD machine always set this value to the maximum (65535 bytes). Specifically, Test Cases 1 and 3 included options in the TCP headers of the packets sent to the target host and the responses received from the Windows machines had the window size value set to 17280, the Linux machines set that value to 5712, and the FreeBSD set the value to 65535. In Test Cases 9, 13, and 15, there were no options included in the packet sent and the responses received from the Windows machines had the window size value set to 17080, the Linux machines set that value to 5760, and the FreeBSD again set the value to 65535.

b. Options

The options field is another key point to differentiate OSes. Not all OSes support all options and nor do they list the supported options in the same order. Windows machines include in their option field only the "Maximum Segment Size", where other OSes may include more options. Furthermore, it was observed that though some OSes support the same options and present them in the same order, they may support different values for the window scale, as happens in the case of Fedora Core 4 and Red Hat Linux 9.0.

c. No Reply

There are some cases where the RFCs do not specify clearly the appropriate response from a host receiving some types of packets. Thus, the developers of the TCP/IP stack may implement different behavior for their OS. Examples of these

cases are the test cases of the NULL packet, the FIN packet, and the PSH packet described in Test Cases 2, 11, and 14, respectively. In those cases, only the Windows machines sent back a response.

d. Hop Limit Value

As in the case of IPv4, the Windows OSes always set this value to 128 and the other OSes always use the value 64. As described with respect to IPv4, the hop limit value can only be used to exclude OSes. For example, if the hop limit value in a received packet is more than 64 that host could not be using an OS that sets the initial hop limit value to 64 or less.

e. Traffic Class

In the test cases presented earlier, the traffic class field of the packet sent to the target machine was set to a randomly selected value. However, many different values were tested but they resulted in the same responses. From the results received, it seems that only the FreeBSD 6.0 and the MacOS X set this field to a value other than the default, which is "0." It was observed that there are cases where FreeBSD may use 0x0, 0x3, 0x8, 0xD or 0xe for the responses sent back. MacOS X set this value to "0" when the response was to accept a connection with the other machine (SYN, ACK packet) and to the value extracted from the received packet in all other cases. Although this value seems to give some clue about the OS, actually it is not safe to use. This is because RFC 2460 mandates, "An upper-layer protocol must not assume that the value of the Traffic Class bits in a received packet are the same as the value sent by the packet's source," because any intermediate node could possibly change this value. So it would be wise, at least for now, to not use it for OS fingerprinting because this field is still under experimental use.

f. Flow Label

Similar to the traffic class field, the value used for the flow label field was selected randomly. While many different values were tested no difference was observed in the responses. The common response is for all OSes to set this value to "0." However, FreeBSD and MacOS X set this field to a random value when they were accepting a request to establish a connection. MacOS X set this field to the same value as that in the packet received for all other cases. The only case that was observed where this value was

set to “0” by MacOS was in the case of the ICMPv6 “port unreachable” message. Possibly the other OSes do not support this functionality and simply set this value to “0” as allowed by RFC 2460.

g. Payload

The payload value counts the number of bytes included in the data field of the IPv6 packet. Thus, it is closely related to the amount of TCP options included in the packet. Only when the amount of options carried in a TCP segment varies, it is possible to make a guess (but not accurate) about the type of OS. In the test cases described earlier, we can characterize an OS belonging to a family of OSes like Windows or Linux.

h. ICMPv6 port Unreachable

In the IPv4 case, it was observed that the amount of original data sent back from the OSes was not constant, and this was a factor for identifying the OS. In the case of IPv6, the ICMP protocol has been replaced from the ICMPv6, which is defined in RFC 2463. The “port unreachable” message is identified as a Type 1 Code 4 message. In Test Case 8, a UDP packet sent to a closed port included some data. The OSes sent back an ICMPv6 “port unreachable” response but the amount of original data sent back never exceeded 1184 bytes, regardless of the amount of data in the original packet. Further, this behavior was observed for all OSes. The point here is that it is not possible to identify the OS based on the amount of original data included in the ICMPv6 header.

The two tools explored by this thesis were effective in providing clues to the underlying OS used on a target machine regardless of the IP version encountered as the tools use information carried in the transport and Network layer headers. We next look to see if the changes in the IPv6 header format open new opportunities to extract information that may lead to the identity of the OS.

V. OS DETECTION METHODS ENABLED BY IPV6

A. OVERVIEW

The research of this thesis so far was concentrated on the current methods for OS fingerprinting under IPv4 and the feasibility to use these methods also with IPv6. In the previous chapter, it was shown that the existing methods for OS fingerprinting on IPv4 could possibly be used on IPv6. However, the necessary changes need to be made on the IP header of the packet so that it conforms to the requirements of the new protocol. IPv6, however, introduces a new concept in the overall protocol stack architecture: the extension headers.

1. Optional Information In IPv6

In IPv4, there was a variable-length optionfield that the originator of the packet used to request specific handling for the packet by the network or the receiver. In IPv6, this field is no longer available. The IPv6 header is always 40 bytes long. However, optional Internet layer information may be encoded in separate headers that may be placed between the IPv6 header and the upper layer header. There is a small number of such extension headers defined, each identified by a distinct Next Header value. A full implementation of IPv6 includes the following extension headers:

- Hop-by-Hop options
- Routing
- Fragment
- Destination options
- Authentication
- Encapsulation Security Payload

With exception of the Hop-by-Hop extension header, these headers are not examined or processed by any node along a packet's delivery path, until the packet reaches the node identified in the Destination Address field of the IPv6 header. The Hop-

by-Hop options header is examined and processed by every node along a packet's delivery path. Each of these extension headers is subject to specific format requirements. The first four of them are specified in RFC 2460 and the last two in RFCs 2402 and 2406, respectively.

2. Research Concept

This chapter concentrates on the possibility of OS fingerprinting methods using any of the extension headers. The idea behind the use of the extension headers in order to detect the OS is to identify possible inconsistencies within the guidance provided by the RFCs in the way the OSes respond to received packets. In a departure from the methods described in the previous chapter, we do not observe the type of information the target OS will include in the IPv6 or TCP header. This has been already examined. Also, the objective of the extension headers is to request some type of service from the network or the receiver of the packet. In this case, the network or the receiver could possibly provide the requested type of service or even better, at least for our purposes, would not understand the requested type of service and so will respond with an ICMPv6 packet pointing to the unrecognized value. If the requested type of service is understood and supported by the receiver, it will handle the packet in the appropriate way and proceed to the next header, which is either TCP or UDP. That means that we will not receive back a packet with any of the extension headers included. Another reason for receiving an ICMPv6 response from the target OS is the case where we craft and send to the target OS a packet with an extension header but with some invalid values included in the fields of the extension headers. For those cases, RFCs define the appropriate response but it may be possible that the target OS will not respond with the expected response. The problem with this case is that if the extension header will be examined or processed by any intermediate router, it may be possible for the packet to be dropped and never reach its destination. So, it is important to ensure that the packet we craft and send to the target OS will get through the network and reach the destination.

B. OS FINGERPRINTING METHODS ENABLED BY IPV6 EXTENSION HEADERS

The following extension headers were examined for utility in identifying the remote OS. Some key identifying factors are described below:

1. Routing Header

The routing header is very similar to IPv4's Loose Source and Record Route option. This header is used by the originator of the packet to list one or more intermediate nodes that must be visited on the way to the destination. The format of the Routing Header, as it is specified in RFC 2460, is presented in Figure 14 below.

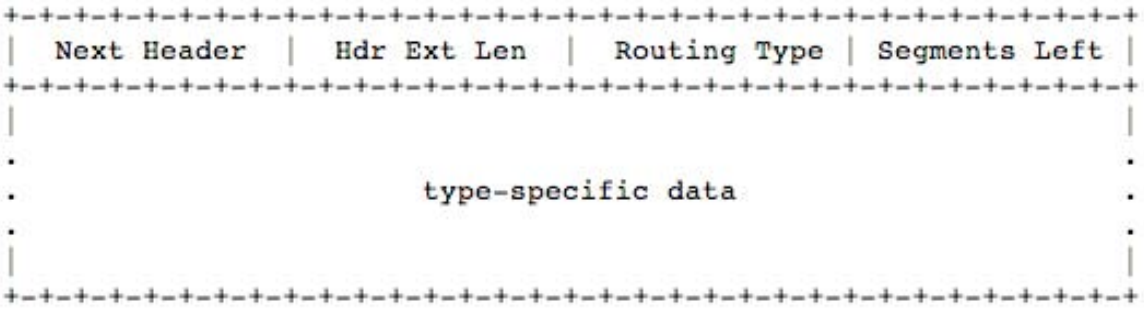

Figure 14. Routing header format.

- Next header: 8-bit selector. Identifies the type of header immediately following the routing header.
- Hdr Ext length: 8-bit unsigned integer. Length of the routing header in 8-octet units, not including the first 8 octets.
- Routing type: 8-bit identifier of a particular routing header variant. In RFC 2460, only the routing type "0" is described. Other routing types supported or experimental are presented in the Table 35 below.
- Segments left: 8-bit unsigned integer. Number of route segments remaining, i.e. number of explicitly listed intermediate nodes still to be visited before reaching the final destination.
- Type-specific data: variable-length field, the format of which is determined by the routing type, and of length is such that the complete routing header is an integer multiple of 8 octets long.

0	Source Route	[IPV6]
1	Nimrod	[CHARLES LYNN]
2	Type 2 Routing Header	[RFC3775]
253	RFC3692-style Experiment 1 (*)	[RFC-fenner-iana-exp-2780-05.txt]
254	RFC3692-style Experiment 2 (*)	[RFC-fenner-iana-exp-2780-05.txt]

Table 35. Routing types[16].

For the routing header, the following methods were found that could possibly be used as a fingerprint for the target OS.

a. Test case 1, Unrecognized routing type

This method sends a packet with a routing extension header following the IPv6 header and, optionally, the TCP header following the routing header as the upper layer protocol. The routing type field is set to an unrecognized value such as "0XFF," the segments left field is set to "1", and the data field set to the "::0" address. As it is specified in RFC 2460, when the receiver of the packet, while processing the routing header, encounters an unrecognized type it will make the following decision based on the value of the segments left field. If the segments left field is "0", then the receiver will ignore the routing header and proceed to process the next header. Otherwise, the receiver will discard the packet and send back an ICMPv6 Type 4, Code 0 (parameter problem, erroneous header field encountered) message to the originator of the packet pointing to the unrecognized routing type. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 37.

b. Test case2, Unrouted address

This method sends a packet with a routing extension header following the IPv6 header. The routing type field is set to the default type "0", the segments left field is set to "1", and the data field set to the ::0 address. As it is specified in RFC 2460, the first node to be visited is the address specified in the destination address field in the IPv6 header. When the packet reaches that node, then the same node will have to route the packet through the next address listed in the type-specific data field[12]of the routing extension header. In this case, that address is considered unrouted, thus the node would not be able to forward the packet. The appropriate response for this case is not explicitly

[12] A list with the addresses to be visited before the packet reaches the final destination

defined in RFC 2460 and the responses by different OSes may vary. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 38.

c. Test case 3, Incorrect extension header length

This method also sends a packet with a routing extension header following the IPv6 header and, optionally, the TCP header following the Routing header, as the upper layer protocol. The routing type field is set to the default type "0", the segments left field is set to "2", and the data field contains the ::0 address. As specified in RFC 2460, when the receiver of the packet, while processing the routing header, determines that the segments left field is greater than the routing addresses in the routing extension header, it should discard the packet and send an ICMPv6 Type 4, Code 0 (parameter problem, erroneous header field encountered) message to the originator of the packet. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 39.

2. Destinations Options Header

The destination options header is used to carry optional information that needs to be examined only by a packet's destination node. The format of the destination options header, as it is specified in RFC 2460, is presented in Figure 15 below.

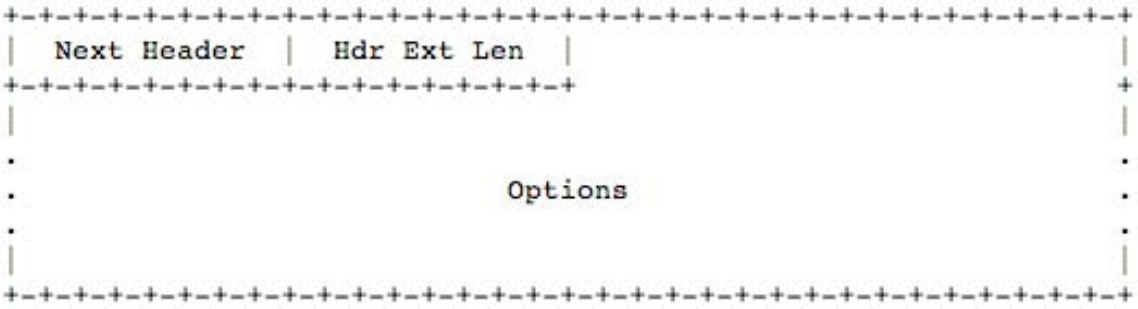

Figure 15. Destinations options header.

- Next Header: 8-bit selector. Identifies the type of header immediately following the Destination Options header.

- Hdr Ext Len: 8-bit unsigned integer. Length of the Destination Options header in 8-octet units, not including the first 8 octets.
- Options: Variable-length field, of length such that the complete destination options header is an integer multiple of 8 octets. Contains one or more Type-Length-Value (TLV) encoded options, as described in RFC 2460 section 4.2. The format of this field is presented in Figure 16 below.

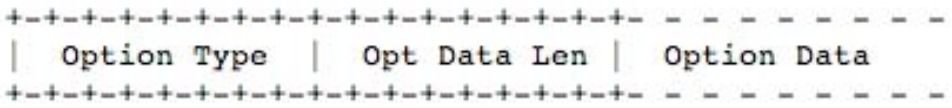

Figure 16. TLV encoded options format.

- Option Type: 8-bit identifier of the type of option. In RFC 2460, only the Pad1 and PadN options are described. Other supported options are experimental as presented in Table 36 below.
- Opt Data Len: 8-bit unsigned integer. Length of the Option Data field of this option, in octets.
- Option Data: Variable-length field. Option-Type-specific data.

In the destination header, one or more options can be included. Each one requires a separate one of the TLV encoded options.

HEX	act	chg	rest		
0	00	0	00000	Pad1	[IPV6]
1	00	0	00001	PadN	[IPV6]
C2	11	0	00010	Jumbo Payload	[JUMBOGRAM]
C3	11	0	00011	Unassigned	
4	00	0	00100	Tunnel Encapsulation Limit	[TUNNEL]
5	00	0	00101	Router Alert	[RFC 2711]
C9	11	0	01001	Home Address	[RFC3775]
8A	10	0	01010	Endpoint Identification	[CHARLES LYNN]
0x1e	00	0	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0x3e	00	1	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0x5e	01	0	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0x7e	01	1	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0x9e	10	0	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0xbe	10	1	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0xde	11	0	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]
0xfe	11	1	11110	RFC3692-style Experiment (*)	[RFC-fenner-iana-exp-2780-05.txt]

Table 36. Supported option types [16].

For the Destinations Options header the following method was found that could possibly be used to fingerprint a target OS.

a. Test case 4, Unrecognized destination type

This method sends a packet with a destination option extension header following the IPv6 header and, optionally, the TCP header following the routing header, as the upper layer protocol. The Destination Type is set to an unrecognized value for a Destination Header. In this case it was set to 0XC2, which is for the “Jumbo Payload,” which is an optional type supported only by the hop-by-hop extension header. The option type codes are internally encoded such that the their highest order two bits specify the action that must be taken if the processing IPv6 node does not recognized the option type. In this case, the expected response is to “discard the packet and, only if the packet's Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type.” [17]. A summary of the packet sent to each host in the network and the responses received from them is presented in Table 40.

Test Case 1

Unrecognized routing type

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	44/24[13]
	NEXT HEADER	0X2B (ROUTING)
	HOP LIMIT	64
ROUTING	NEXT HEADER	6 (TCP) 59(NO NEXT HEADER)
	LENGTH	2
	TYPE	0XFF (UNRECOGNIZED TYPE)
	SEGMENT LEFT	1
	ADDRESS	::0
TCP	SEQUENCE	ISN
	ACKNOWLEDGE	0
	FLAGS	SYN
	HEADER	20
	OPTIONS	--

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	0	0
	PAYLOAD	92/72	92/72	92/72	92/72	92/72	92/72	92/72	92/72
	HOP LIMIT	128	128	128	64	64	64	64	64
ICMPv6	TYPE	4	4	4	4	4	4	4	4
	CODE	0	0	0	0	0	0	0	0
	POINTER	0X2A	0X2A	0X2A	0X2A	0X2A	0X2	0X2A	0X2A
			PARAMETER PROBLEM MESSAGE ERRONEOUS HEADER FIELD ENCOUNTERED						

Table 37. Test case 1, Unrecognized routing type.

[13] 44 Bytes when there is TCP header and 24 Bytes when there is no TCP header following the Routing extension header

Test Case 2

Unrouted address

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	44/24
	NEXT HEADER	0X2B (ROUTING)
	HOP LIMIT	64
ROUTING	NEXT HEADER	6 (TCP) 59(NO NEXT HEADER)
	LENGTH	2
	TYPE	0
	SEGMENTS LEFT	1
	ADDRESS	::0
TCP	SEQUENCE	ISN
	ACKNOWLEDGE	0
	HEADER	20
	FLAGS	SYN
	OPTIONS	--

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	NO REPLY	NO REPLY	NO REPLY	NO REPLY	NO REPLY
	FLOW LABEL	0	0	0					
	PAYLOAD	92	92	92					
	HOP LIMIT	128	128	128					
ICMPv6	TYPE	4	4	4					
	CODE	0	0	0					
	POINTER	0x30	0x30	0x30					
				PARAMETER PROBLEM MESSAGE ERRONEOUS HEADER FIELD ENCOUNTERED					

Table 38. Test case 2, Unrouted Address.

Test Case 3

Incorrect extension header length

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	24
	NEXT HEADER	0X2B (ROUTING)
	HOP LIMIT	64
ROUTING	NEXT HEADER	59(NO NEXT HEADER)
	LENGTH	1
	TYPE	0
	SEGMENTS LEFT	2
	ADDRESS	::0

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	0	0
	PAYLOAD	72	72	72	72	72	72	72	72
	HOP LIMIT	128	128	128	64	64	64	64	64
ICMPv6	TYPE	4	4	4	4	4	4	4	4
	CODE	0	0	0	0	0	0	0	0
	POINTER	0X29	0X29	0X29	0X29	0X29	0X1	0X29	0X29

Table 39. Test case 3, Incorrect extension header length.

Test Case 4

Unrecognized destination type

Packet send

IPV6	TRAFFIC CLASS	X
	FLOW LABEL	Y
	PAYLOAD	28/8
	NEXT HEADER	0X3C (DESTINATIONS)
	HOP LIMIT	64
DESTINATION	NEXT HEADER	6 (TCP) 59(NO NEXT HEADER)
	LENGTH	0
	TYPE	0XC2 (JUMBO PACKET)
TCP	SEQUENCE	ISN
	ACKNOWLEDGE	0
	FLAGS	SYN
	HEADER	20
	OPTIONS	--

Packet received		Windows Server 2003 Enterprise Edition SP1	Windows Server 2003 Standard Edition	Windows XP Pro SP2	Red Hat Enterprise Linux 4 WS	FEDORA CORE 4	Red Hat Linux 9.0	FreeBSD 6.0	MAC OS X
IPV6	TRAFFIC CLASS	0	0	0	0	0	0	0	0
	FLOW LABEL	0	0	0	0	0	0	0	0
	PAYLOAD	76	76	76	76	76	76	76	76
	HOP LIMIT	128	128	128	64	64	64	64	64
ICMPv6	TYPE	4	4	4	4	4	4	4	4
	CODE	0	0	0	2	2	2	2	2
	POINTER	0X2A	0X2A	0X2A PARAMETER PROBLEM MESSAGE ERRONEOUS HEADER FIELD ENCOUNTERED	0X2A PARAMETER PROBLEM MESSAGE UNRECOGNIZED IPV6 OPTION ENCOUNTERED	0X2A	0X2A	0X2A	0X2A

Table 40. Test case 4, Unrecognized destination type.

C. ANALYSIS

From the results above, it is evident that not all OSes respond with the same values in their headers and there are cases where their behavior is different. The common characteristic of the methods described in the test cases earlier is that all of them trigger the target host to respond with an ICMPv6 message. Although the RFCs provide explicit guidance as to the appropriate response for most of the cases, the OSes do show some deviation from that guidance. The various differences that were explored in the previous chapter regarding the IPv6 header appear in these cases also and are not discussed again. However, the differences that are identified by examining the responses in the ICMPv6 headers are discussed below.

1. ICMPv6 Pointer Value

The pointer value placed in an ICMPv6 message sent by a node points to the unrecognized field of the original packet received by that node. In Test Cases 1 and 3, almost all OSes sent an ICMPv6 message Type 4 Code 0 to the originator of the packet pointing to the 0x2A and 0x29 respectively, which are the routing type and segments left fields of the original packet. An exception was observed for Red Hat Linux 9.0, which includes different values for the pointer fields, 0x2 and 0x1 respectively. It appears that Red Hat Linux 9.0 still points to the same unrecognized fields but starts the counting of the bytes in the original packet from the first byte of the routing extension header, offsetting it by 40 bytes compared with the values inserted by the other OSes.

2. No Reply

In Test Case 2, the address included in the routing header that is supposed to be visited on the path to reach the destination node cannot be routed. This case is not explicitly covered in RFC 2460 and we observe that only the Windows machines respond with an ICMPv6 message Type 4, Code 0 (parameter problem erroneous header field encountered) pointing to the 0x30 nyte of the original packet. This byte is part of the IPv6 address included in the routing header, which we set to “::0”. It is actually the first byte after the initial 64-bit prefix. This is because the smallest prefix in an IPv6 address is 64 bits and we set it to “0.” When the node examines the prefix value and finds that it is zero, it determines that it cannot process the packet and sends back the error message to the source address.

3. ICMPv6 Code Value

In Test Case 4, we observe a slightly different response between the Windows machines and the rest of the OSes. Windows responds with an ICMPv6 Type 4, Code 0 and the rest of the OSes send back an ICMPv6 Type 4, Code 2 message. This is an example of what can be interpreted differently by different vendors of the OSes. It is evident in both cases that this packet can not be processed and the problem is at the destination option type 0XC2 (jumbo payload). The jumbo payload option is supported only by the hop-by-hop extension header, so Windows machines on the one hand assume that the 0XC2 value is erroneous for this extension header (destination header) and the other OSes, on other hand, consider this value as unrecognized.

It can be concluded, then, that the extension headers may provide opportunities for fingerprinting the target host OS. While this list of techniques indicate the fertility of IPv6 extension headers for eliciting information about the OSes employed on a network, it is not comprehensive as other techniques may be used which were not explored in this thesis

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

The objective of OS fingerprinting is to identify the OS type a target machine is running from a remote machine. OS fingerprinting is feasible because developers of different OSes may interpret the guidance provided by the RFCs differently, and consequently their network protocol stack implementation may generate responses bearing unique markers to certain probing packets. The key part of OS fingerprinting is the finding of suitable probing packets for different OSes. Effective OS fingerprinting tools have been developed for probing hosts running the IPv4 protocol stack. This thesis has shown that the methods used by these tools can also be used for probing a host that runs an IPv6 protocol stack and that IPv6 extension headers could enable additional methods for OS fingerprinting.

A. CONCLUSIONS

Tables 41 and 42 below summarize the results presented in Chapter IV and V. They indicate the effectiveness of each method evaluated in this thesis, in terms of its ability to fingerprint the OS type of a host known to run an IPv6 protocol stack.. For example, by applying the first TCP/UDP based method on the selected set of OSes we have identified five different fingerprints, each one associated with a particular OS or a set of OSes. Note that after a brief description of each method are two page numbers: one pointing to where the detail of the probing packet is given and the other pointing to where the detail of the response packets from different OSes is presented.

Methods Tested		Sets of OSes with Unique Fingerprint
TCP/UDP based	SYN, ECN packet with options to an open port [cf. pp 73; pp 76]	A. MS Server 2003 Enterprise/Standard, Windows XP Pro B. Red Hat Enterprise Linux 4 WS, Fedora Core 4 C. Red Hat Linux 9.0 D. FreeBSD 6.0 E. MAC OS X
	NULL packet with options to an open port [cf. pp 74; pp 77]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0, MAC OS X
	FIN, SYN, PSH, URG packet with options to an open port [cf. pp 74; pp 78]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4 3. Red Hat Linux 9.0 4. FreeBSD 6.0 5. MAC OS X
	ACK packet with options to an open port [cf. pp 74; pp 79]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0 3. MAC OS X

	SYN packet with options to a closed port [cf. pp 74; pp 80]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0 3. MAC OS X
	ACK packet with options to a closed port [cf. pp 74; pp 81]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0 3. MAC OS X
	ACK packet with options to a closed port [cf. pp 75; pp 82]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0 3. FreeBSD 6.0 4. MAC OS X
	UDP packet with data to a closed port [cf. pp 75; pp 83]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0, MAC OS X

	SYN packet without options to an open port [cf. pp 84; pp 86]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0 3. FreeBSD 6.0, MAC OS X
	SYN, ACK packet without options to an open port [cf. pp 84; pp 87]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0 3. MAC OS X
	FIN packet without options to an open port [cf. pp 84; pp 88]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0, MAC OS X
	FIN, ACK packet without options to an open port [cf. pp 84; pp 89]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0 3. MAC OS X

	SYN, FIN packet without options to an open port [cf. pp 84; pp 90]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0 3. FreeBSD 6.0 4. MAC OS X
	PSH packet without options to an open port [cf. pp 84; pp 91]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0, FreeBSD 6.0, MAC OS X
	SYN, ECN, CWR packet without options to an open port [cf. pp 84; pp 92]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, Red Hat Linux 9.0 3. FreeBSD 6.0, MAC OS X

Table 41. Consolidated results from using UDP/TCP methods.

Methods Tested		Sets of OSes with Unique Fingerprint
IPv6 extension headers based	Unrecognized routing type [cf. pp 99; pp 103]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Fedora Core 4, FreeBSD 6.0, MAC OS X 3. Red Hat Linux 9.0
	Unrouted address [cf. pp 99; pp 104]	1. MS Server 2003 Enterprise/Standard, Windows XP Pro 2. Red Hat Enterprise Linux 4 WS, Red Hat Linux 9.0, Fedora Core 4, FreeBSD 6.0, MAC OS X
	Incorrect extension header Length [cf. pp 100; pp 105]	A. MS Server 2003 Enterprise/Standard, Windows XP Pro B. Red Hat Enterprise Linux 4 WS, Fedora Core 4, FreeBSD 6.0, MAC OS X C. Red Hat Linux 9.0
	Unrecognized destination type [cf. pp 102; pp 106]	A. MS Server 2003 Enterprise/Standard, Windows XP Pro B. Red Hat Enterprise Linux 4 WS, Red Hat Linux 9.0, Fedora Core 4, FreeBSD 6.0, MAC OS X

Table 42. Consolidated results from using IPv6 extension header.

The following conclusions can be drawn from the results summarized in the tables above:

- Tools developed in the IPv4 environment can be used to fingerprint an IPv6 host effectively. They are able to differentiate a majority of the OSes in the selected test set. However, they can no longer be used to distinguish XP Pro from the other two server versions of Microsoft Windows. This might be because all three versions of Windows are bundled with the same IPv6 code. The confirmation is left for further study.
- The IPv6 extension header based methods seem not as effective as the UDP/TCP based methods. The methods tried for this thesis trigger the same responses from Red Hat Enterprise Linux 4 WS, Fedora Core 4, FreeBSD 6.0, and MAC OS X. Again, it might be because all these OSes have borrowed the same code base. More work is required in this area to either confirm this conjecture or develop more effective fingerprinting probes.
- None of the methods tried in this thesis can distinguish between Red Hat Enterprise 4 WS and Fedora Core 4. This is true even in the IPv4 environment.

Another point of significant concern for OS fingerprinting methods is the lack of tools for crafting IPv6 packets. IPv6 is still experimental and there are not many tools available with an easy-to-use interface. In this thesis SmartBits 6000C was used. There were, however, cases where the crafting had to be done manually and the appropriate values provided in hexadecimal format. This is a very time consuming process because a single incorrect value may make the packet unusable or un-routable.

B. FUTURE RESEARCH

An important aspect that is crucial for OS fingerprinting is the size of the database holding the known fingerprints. A larger database would lead to more accurate inferences about the target OS. Thus, it is important to build a database with as many fingerprints as possible. Each OS explored should be added to the database. It was noted that Nmap includes about 1,500 fingerprints in its database. From our study we

concluded that no single method provides definitive fingerprint for all OSes in an IPv6 environment. An application could be developed to exploit all these methods and maintain a fingerprint database specifically for IPv6 hosts, iteratively applying probes to refine each inference and so automate the process of recognizing an OS.

Another limitation of this work is that only a few extension headers of IPv6 were evaluated. This is largely due to the fact that many of the supported options are still experimental and thus they are not fully defined. It should be beneficial to revisit this issue when the specifications of IPv6 extension headers become more concrete and more stable.

Finally, not all of the methods described in Chapters IV and V were effective. However, we should not conclude that these methods are absolutely without merit. The OSes sample used is not large enough to be considered exhaustive. These methods may be more successful against other OSes. Thus, one possible way to extend this research could be the application of the methods described in this thesis to additional OSes so that we can have a better idea about which methods are useful and which are not.

LIST OF REFERENCES

1. James F. Kurose, Keith W. Ross. Computer Networking; A TOP DOWN APPROACH FETURING THE INTERNET 3rd edition. 2004

2. James F. Kurose, Keith W. Ross. Computer Networking; A TOP DOWN APPROACH FETURING THE INTERNET 3rd edition, pp 344. 2004

3. Solensky 1996

4. Stephen E. Deering, Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, pp 2.

5. Stuart McClure, Joel Scambray, George Kurtz. Hacking Exposed Network Security & Solutions 5th Edition, p 69

6. Fyodor. "Remote OS detection via TCP/IP Stack Fingerprinting". 18 Oct. 1998. <www.insecure.org>.

7. Stuart McClure, Joel Scambray, George Kurtz. Hacking Exposed Network Security & Solutions 5th Edition, p 52

8. Stuart McClure, Joel Scambray, George Kurtz, Hacking Exposed Network Security & Solutions 5th Edition, p 79

9. Stuart McClure, Joel Scambray, George Kurtz. Hacking Exposed Network Security & Solutions 5th Edition, p 73

10. Stuart McClure, Joel Scambray, George Kurtz. Hacking Exposed Network Security & Solutions 5th Edition, p 68

11. Fyodor. "Remote OS detection via TCP/IP stack Fingerprinting". 18 Oct. 1998. <www.insecure.org/nmap/nmap-fingerprinting-article.html>.

12. Thomas Ptacek, Tim Newsham; "Insertion, Evasion and Denial of service: Eluding Network Intrusion Detection". <www.clark.net/<126>roesch/idspaper.html>.

13. Stuart McClure, Joel Scambray, George Kurtz, Hacking Exposed Network Security & Solutions 5th Edition, p 69

14. Philosophe.com. A thoughtful approach to web site quality. 7 Jun 1999. <http://www.philosophe.com/audience/operating_systems.html>.

15. www.w3schools.com.Browser statistics. 16 Jun 2006. <http://www.w3schools.com/browsers/browsers_stats.asp>.

16. IANA. “IP VERSION 6 PARAMETERS”. 25 Jul 2006. www.iana.org/assignments/ipv6-parameters.

17. Stephen E. Deering, Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, pp 8.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Neal Ziring
   National Security Agency
   Fort George G. Meade, Maryland

4. Matthew N. Smith
   National Security Agency
   Fort George G. Meade, Maryland